MIDAS: Finding the Right Web Sources to Fill Knowledge Gaps

Anonymous authors

Abstract

Knowledge bases, massive collections of facts (RDF triples) on diverse topics, support vital modern applications. However, existing knowledge bases contain very little data compared to the wealth of information on the Web. This is because the industry standard in knowledge base creation and augmentation suffers from a serious bottleneck: they rely on domain experts to identify appropriate web sources to extract data from. Efforts to fully automate knowledge extraction have failed to improve this standard: these automated systems are able to retrieve much more data and from a broader range of sources, but they suffer from very low precision and recall. As a result, these large-scale extractions remain unexploited.

In this paper, we present Midas, a system that harnesses the results of automated knowledge extraction pipelines to repair the bottleneck in industrial knowledge creation and augmentation processes. Midas automates the suggestion of good-quality web sources and describes what to extract with respect to augmenting an existing knowledge base. We make three major contributions. First, we introduce a novel concept, web source slices, to describe the contents of a web source. Second, we define a profit function to quantify the value of a web source slice with respect to augmenting an existing knowledge base. Third, we develop effective and highly-scalable algorithms to derive high-profit web source slices. We demonstrate that Midas produces high-profit results and outperforms the baselines significantly on both real-word and synthetic datasets.

1. Introduction

Knowledge bases support a wide range of applications and enhance search results for multiple major search engines, such as Google and Bing [2].The coverage and correctness of knowledge bases are crucial for the applications that use them, and for the quality of the user experience. However, there exists a gap between facts on the Web and in knowledge bases: compared to the wealth of information on the Web, most knowledge bases are largely incomplete, with many facts missing. For example, one of the largest knowledge bases, Freebase [9, 1], does not provide sufficient facts for *different types of cocktails* such as *the ingredients of Margarita*. Yet, such information is explicitly profiled and described by many web sources, such as *Wikipedia*.

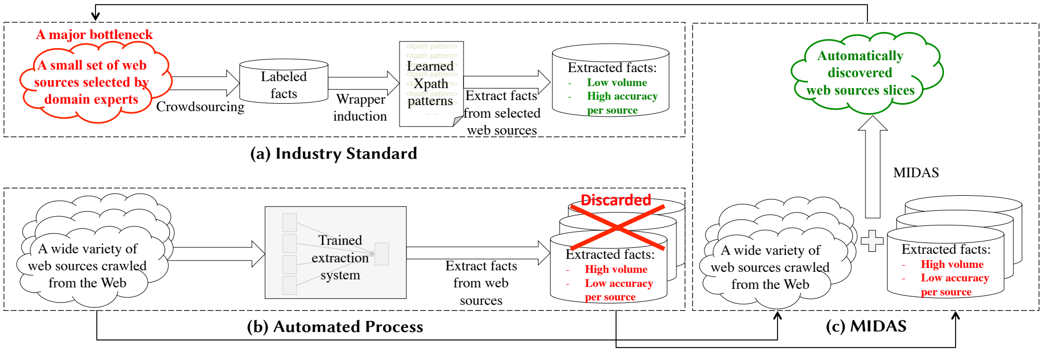

A major bottleneck
A small set of web sources selected by domain experts
Crowdsourcing
Labeled facts
Wrapper induction
Learned Xpath patterns
Extract facts from selected web sources
Extracted facts:
- Low volume
- High accuracy per source
(a) Industry Standard
A wide variety of web sources crawled from the Web
Trained extraction system
Extract facts from web sources
Discarded
Extracted facts:
- High volume
- Low accuracy per source
(b) Automated Process
Automatically discovered web sources slices
MIDAS
A wide variety of web sources crawled from the Web
Extracted facts:
- High volume
- Low accuracy per source
(c) MIDAS

Figure 1: Two knowledge extraction procedures and Midas. The output of the automated process (b) is often discarded in industrial production due to low accuracy. Midas uses the the automatically-extracted facts to identify the right web sources for the semi-automated process under the industry standard and therefore resolves a major bottleneck.

Industry standard. Industry typically follows a semi-automated knowledge extraction process to create or augment a knowledge base with facts that are new to an existing knowledge base (or new facts) from the Web. This process (Figure 1a) first relies on domain experts to select web sources; it then uses crowdsourcing to annotate a fraction of entities and facts and treats them as the training data; finally, it applies wrapper induction [21, 23] and learns Xpath patterns to extract facts from the selected web sources. Since source selection and training data preparation are carefully curated, this process achieves high precision and recall with respect to each selected web source. However, it can only produce a small volume of facts overall and cannot scale, as the source-selection step is a severe bottleneck, relying on manual curation by domain experts.

Automated process. To conquer the scalability limitation in the industry standard, automated knowledge extraction [15, 32] attempts to extract facts with little or no human intervention. Instead of manually selecting a small set of web sources, automated extraction (Figure 1b) often takes a wide variety of web sources, e.g., ClueWeb09 [12], as input and uses facts in an existing knowledge base, or a small portion of labeled input web sources, as training data. This automated extraction process is able to produce a vast number of facts. However, because of the limited training data (per source), especially for uncommon facts, e.g., *the ingredients of Margarita*, this process suffers from low accuracy. The TAC-KBP competition showed that automated processes [35, 5, 36, 14] can hardly achieve above 0.3 recall, leaving a lot of the wealth of web information unexploited. Due to this limitation, such automatically extracted facts are often abandoned for knowledge bases in industrial production.

In this paper, we propose Midas[1], a system that harnesses the correct[2] extractions of the *automated process* to automatically identify suitable web sources and repair the bottleneck in the *industry standard*. The core insight of Midas is that the automatically extracted facts, *even though they may not be of high overall accuracy and coverage, give clues about which web sources contain a large amount of valuable information, allow for easy annotation, and are worthwhile for extraction.* We demonstrate this through an example.

[1] Our system is named after King Midas, known in Greek mythology for his ability to turn what he touched into gold.

[2] We refer to correct facts as facts with confidence value ≥ 0.7 as true.

ID	subject	predicate	object	new?	web source
t_1	Project Mercury	category	space_program	N	http://space.skyrocket.de/doc_sat/mercury-history.htm
t_2	Project Mercury	started	1959	N	http://space.skyrocket.de/doc_sat/mercury-history.htm
t_3	Project Mercury	sponsor	NASA	N	http://space.skyrocket.de/doc_sat/mercury-history.htm
t_4	Project Gemini	category	space_program	N	http://space.skyrocket.de/doc_sat/gemini-history.htm
t_5	Project Gemini	sponsor	NASA	N	http://space.skyrocket.de/doc_sat/gemini-history.htm
t_6	Atlas	category	rocket_family	Y	http://space.skyrocket.de/doc_lau_fam/atlas.htm
t_7	Atlas	sponsor	NASA	Y	http://space.skyrocket.de/doc_lau_fam/atlas.htm
t_8	Atlas	started	1957	Y	http://space.skyrocket.de/doc_lau_fam/atlas.htm
t_9	Apollo program	category	space_program	N	http://space.skyrocket.de/doc_sat/apollo-history.htm
t_{10}	Apollo program	sponsor	NASA	N	http://space.skyrocket.de/doc_sat/apollo-history.htm
t_{11}	Castor-4	category	rocket_family	Y	http://space.skyrocket.de/doc_lau_fam/castor-4.htm
t_{12}	Castor-4	started	1971	Y	http://space.skyrocket.de/doc_lau_fam/castor-4.htm
t_{13}	Castor-4	sponsor	NASA	Y	http://space.skyrocket.de/doc_lau_fam/castor-4.htm

Figure 2: Facts that are correctly extracted from `http://space.skyrocket.de`. We compare the extracted facts with Freebase and mark the facts that are absent from Freebase as "Y" in the "new" column.

Example 1. *Figure 2 shows a snapshot of high-confidence facts (subject, predicate, object) extracted from 5 web pages under web domain* `http://space.skyrocket.de`*. Automated extraction systems may not be able to obtain high precision and recall in extracting facts from this website due to lack of effective training data. However, the few correct extracted facts give important clues on what one could extract from this site.*

For each fact, the subject indicates an entity; the predicate and object values further describe properties associated with the entity. For example, fact t_1 specifies that the category property of the entity Project Mercury *is* space program. *Entities can form groups based on their common properties. For example, entity "Project Mercury" and entity "Project Gemini" are both "space programs that are sponsored by NASA".*

The facts labeled "Y" in the "new?" column are absent from Freebase. All of these new facts are under the same sub-domain and are all "rocket families sponsored by the NASA." This observation provides a critical insight: one can augment Freebase by extracting facts pertaining to "rocket families sponsored by NASA" from `http://space.skyrocket.de/doc_lau_fam`*.*

Example 1 shows that one can abstract the contents of a web source through extracted facts: A web source often includes facts of multiple groups of homogeneous entities. Each group of entities forms a particular subset of content in the web source, which we call a **web source slice (or slice)**. The common properties shared by the group of entities not only define, but also describe the slice of facts. For example, it is easy to tell that a slice describes *"rocket families sponsored by NASA"* through its common properties, *"category = rocket family"* and *"sponsor = NASA"*. Moreover, entities in a single web source slice often belong to the same type, e.g., *"rocket families sponsored by NASA"*, and thus share similar predicates. The limited number of predicates in a web source slice simplifies annotation. Our objective is to discover web source slices that (1) contain a sufficient number of facts that are absent from the knowledge base we wish to augment, and (2) their extraction effort does not outweigh the benefit.

However, **evaluating and quantifying the suitability** of a web source slice with respect to these two desired properties is not straightforward. In addition, the number of slices in a single web source often grows exponentially with the number of facts, posing a significant **scalability challenge**. This challenge is amplified by the massive number of sources on the Web, in various genres, languages, and domains. Even a single web domain

Slice description	Web source
Education organizations	*http://www.schoolmap.org/school/*
US golf courses	*https://www.golfadvisor.com/course-directory/2-usa/*
Biology facts	*http://www.marinespecies.org*
Board games	*http://boardgaming.com/games/board-games/*
Skyscraper architectures	*http://skyscrapercenter.com/building*
Indian politicians	*http://www.archive.india.gov.in*

Figure 3: Selected top returns (slices) from Midas targeting the augmentation of Freebase. Midas derived slides using facts extracted from a real-world, large-scale, automated knowledge extraction pipeline (name hidden for anonymity) that operates on billions of web pages. New facts refer to extracted facts that are absent from Freebase.

may contain an extensive amount of knowledge. For example, as of July 2018, there are more than 45 million entries in Wikipedia [3].

Midas addresses these challenges through (1) efficient and scalable algorithms for producing web source slices, and (2) an effective profit function for measuring the utility of slices. In this paper, we first formalize the problem of identifying and describing "good" web sources as an optimization problem and then quantify the quality of web source slices through a *profit* function (Section 2). We then propose an algorithm to generate the high-profit slices in a single web source and design a scalable framework to extend this algorithm for multiple web sources (Section 3). Finally, we evaluate our proposed algorithm on both real-word and synthetic datasets and illustrate that our proposed system, Midas, is able to identify interesting web sources slices in an efficient and scalable manner (Section 4).

Example 2. *We applied* Midas *on AnonSys, a dataset extracted by a comprehensive knowledge extraction system, which includes 810M facts extracted from 218M web sources.* Midas *is able to identify and customize "good" web sources for an existing knowledge base. In Figure 3, we demonstrate the 5 highest-profit slices that* Midas *derived to augment Freebase. The web source slices provide new and valuable information for augmenting the existing knowledge base; in addition, many of these web sources contain semi-structured data with respect to entities in the reported web source slice. Therefore, they are easy for annotation.*

2. Problem Definition

In this section, we first define web source slices (Section 2.1); we then use these abstractions to formalize the problem of slice discovery for knowledge base augmentation (Section 2.2).

2.1 Web Source Slice

Web source. URL hierarchies offer access to web sources at different granularities, such as a web domain (`https://www.cdc.gov`), a sub-domain (`https://www.cdc.gov/niosh`), or a web page (`https://www.cdc.gov/niosh/ipcsneng/neng0363.html`). Web domains often use URL hierarchies to classify their contents. For example, the web domain `https://www.golfadvisor.com` classifies facts for *"golf course in Jamaica"* under the finer-grained URL `https://www.golfadvisor.com/course-directory/8545-jamaica`. The URL hierarchies in these web domains divide their contents into smaller, coherent subsets, providing opportunities to reduce unnecessary extraction effort. For example, the web domain `https://www.cdc.gov` requires significant extraction effort as its contents are varied and spread across

Fact table

EID	subject	category	sponsor	started
e_1	Project Mercury	space_program	{NASA}	{1959}
e_2	Project Gemini	space_program	{NASA}	$\emptyset$
e_3	Atlas	rocket_family	{NASA}	{1957}
e_4	Apollo program	space_program	{NASA}	$\emptyset$
e_5	Castor-4	rocket_family	{NASA}	{1971}

Properties

CID	Property
c_1	(category, space_program)
c_2	(category, rocket_family)
c_3	(started, 1959)
c_4	(started, 1957)
c_5	(started, 1971)
c_6	(sponsor, $NASA$)

Web source slices

SID	Properties	Entities	Facts	Description
S_1	$\{c_1, c_3, c_6\}$	$\{e_1\}$	$\{t_1, t_2, t_3\}$	space programs sponsored by NASA and started in 1959
S_2	$\{c_2, c_4, c_6\}$	$\{e_3\}$	$\{\mathbf{t_6}, \mathbf{t_7}, \mathbf{t_8}\}$	rocket families sponsored by NASA and started in 1957
S_3	$\{c_2, c_5, c_6\}$	$\{e_5\}$	$\{\mathbf{t_{11}}, \mathbf{t_{12}}, \mathbf{t_{13}}\}$	rocket families sponsored by NASA and started in 1971
S_4	$\{c_1, c_6\}$	$\{e_1, e_2, e_4\}$	$\{t_1–t_5, t_9, t_{10}\}$	space programs sponsored by NASA
S_5	$\{c_2, c_6\}$	$\{e_3, e_5\}$	$\{\mathbf{t_6–t_8}, \mathbf{t_{11}–t_{13}}\}$	rocket families sponsored by NASA
S_6	$\{c_6\}$	$\{e_1, e_2, e_3, e_4, e_5\}$	$\{t_1–t_5, \mathbf{t_6–t_8}, t_9, t_{10}, \mathbf{t_{11}–t_{13}}\}$	any projects sponsored by NASA

Figure 4: Fact table, properties, and example slices derived from facts in Figure 2. The facts that are absent from Freebase ($t_6, t_7, t_8, t_{11}, t_{12}$, and t_{13}) are highlighted in green.

too many categories; the sub-domain `https://www.cdc.gov/niosh/ipcsneng` represents lower extraction effort, because its content focuses on "international chemical safety information".

Midas considers web sources at all granularity levels of the URL hierarchy.

Contents of a web source. Facts extracted from a web source typically correspond to many different entities. However, they can share common properties: for example, the entities "Atlas" and "Castor-4" (Figure 2) have the common property of being rocket families sponsored by NASA. We abstract and formalize the content represented by a group of entities as a *web source slice* and define it by the entities' common properties. The abstraction of web source slices achieves two goals: (1) it offers a representation of the content of a web source that is easily understandable by humans, and (2) it allows for the efficient retrieval of all facts relevant to that content.

As described in Example 1, an extracted fact corresponds to an entity and describes properties of that entity. Web source slices, in turn, are defined over a group of entities with common properties. To facilitate this exposition, we organize facts of a web source W in a *fact table* F_W (Figure 4). A row in the fact table contains facts that correspond to the same entity (denoted by the subject).

Definition 3 (Fact table)**.** *Let $\mathcal{T}_W = \{(s, p, o)\}$ be a set of facts, in the form of (**s**ubject, **p**redicate, **o**bject), extracted from a web source W, and n be the number of distinct predicates in $\mathcal{T}_W$ ($n = |\{t.p \mid t \in \mathcal{T}_W\}|$). We define the fact table $F_W(\underline{subject}, pred_1, \ldots, pred_n)$, which has a primary key (subject) and one attribute for each of the n distinct predicates. Each fact $t \in \mathcal{T}_W$ maps to a single, non-empty cell in F_W:*

$$\forall t \in \mathcal{T}_W,\ t.o \in \Pi_{t.p}\sigma_{subject=t.s}(F_W)$$

where Π and σ are the Projection and Selection operators in relational algebra.

Note that cells in F_W may contain a set of values, corresponding to facts with the same subject and predicate. For ease of exposition, we use single values in our examples. We now define properties and web source slices over the fact table F_W.

Definition 4 (Property)**.** *A property $c = (pred, v)$ is a pair derived from a fact table F_W, such that pred is an attribute in F_W and $v \in \Pi_{pred}(F_W)$. We further denote with $\mathcal{C}_W$ the set of all properties in a web source W: $\mathcal{C}_W = \cup_{F_W.pred} \cup_{v \in \Pi_{pred}(F_W)} (pred, v)$*

Figure 4 lists all the properties derived from the fact table of our running example. Midas considers properties where the value is strictly derived from the domain of *pred*: $v \in \Pi_{pred}(F_W)$. Our method can be easily extended to more general properties, e.g., "year > 2000"; however, we decided against this generalization, as it increases the complexity of the algorithms significantly, without observable improvement in the results. In addition, Midas does not consider properties on the subject attribute since in most real-word datasets subjects are typically identification numbers.

Definition 5 (Web Source Slice)**.** *Given a set of facts $\mathcal{T}_W$ extracted from web source W, the corresponding fact table F_W, and the collection of properties $\mathcal{C}_W$, a* web source slice *(or slice), denoted by $S(W)$ (or S for short), is a triplet $S(W) = (C, \Pi, \Pi^*)$, where,*

$C = \{c_1, ..., c_k\} \subseteq \mathcal{C}_W$ *is a set of properties;*

$\Pi = \Pi_{subject}\sigma_{c_1 \wedge ... \wedge c_k}(F_W)$ *is a non-empty set of entities, each of which includes all of the properties in C;*

$\Pi^* = \{(s, p, o) | (s, p, o) \in \mathcal{T}_W, s \in \Pi\}$ *is a non-empty set of facts that are associated with entities in* Π.

Example 6. *Figure 4 demonstrates the fact table (upper-left), properties (upper-right), and slices (bottom) derived from the facts of Figure 2. For example, slice S_6 on property $\{c_6\}$ represents facts for projects sponsored by NASA; slice S_4 on properties $\{c_1, c_6\}$ represents facts for space programs sponsored by NASA.*

Canonical slice. Different slices may correspond to the same set of entities. For example, in Figure 4, the slice defined by $\{c_5, c_6\}$ corresponds to entity e_5, the same as slice S_3, but it has a different semantic interpretation: projects sponsored by NASA and started in 1957. Based on the extracted knowledge, it is impossible to tell which slice is more precise; reporting and exploring all of them introduces redundancy to the results and also significantly increases the overall problem complexity. In Midas, we choose to report *canonical slices*: among all slices that correspond to the same set of entities and facts, the one with the maximum number of properties is a canonical slice.

Definition 7. *A slice $S(W) = (C, \Pi, \Pi^*)$ is a canonical slice if there exists no $S'(W) = (C', \Pi, \Pi^*)$ such that $|C'| \leq |C|$.*

Focusing on canonical slices does not sacrifice generality. The canonical slice is always unique, and one can infer the unreported slices from the canonical slices by taking any subset of a canonical slice's properties and validating the corresponding entities. All six slices in Figure 4 are canonical slices that select at least one fact.

2.2 The Slice Discovery Problem

Definition 8 (Problem Definition)**.** *Let $\mathcal{E}$ be an existing knowledge base, $\mathcal{W} = \{W_1, ...\}$ be a collection of web sources, $\mathcal{T}_W$ be the facts extracted from web source $W \in \mathcal{W}$, and $f(\mathcal{S})$ be an objective function evaluating the profit of a set of slices on the given existing knowledge base $\mathcal{E}$. The web source suggestion problem finds a list of web source slices, $\mathcal{S} = \{S_1, ...\}$, such that the objective function $f(\mathcal{S})$ is* ***maximized****.*

Inspired by solutions in [17, 31], we quantify the value of a set of slices as the *profit* (i.e., gain−cost) of using the set of slices to augment an existing knowledge base. We measure the gain as a function of the number of unique new facts presented in the slices, showing the potential benefit of these facts in downstream applications. We estimate the cost based on common knowledge-base augmentation procedures [15, 27, 32], which contain three steps: crawling the web source to extract the facts, de-duplicating facts that already exist in the knowledge base, and validating the correctness of the newly-added facts. In our implementation, we assume that the gain and cost are linear with respect to the number of (new) facts in all slices. This assumption is not inherent to our methodology, and one can adjust the gain and cost functions.

Definition 9. *Let $\mathcal{S}$ be the set of slices derived from web source W and let $\mathcal{E}$ be a knowledge base. We compute the gain and the cost of $\mathcal{S}$ with respect to $\mathcal{E}$ as $G(\mathcal{S}) = | \cup_{S \in \mathcal{S}} S \setminus \mathcal{E}|$ and $C(\mathcal{S}) = C_{crawl}(\mathcal{S}) + C_{de\text{-}dup}(\mathcal{S}) + C_{validate}(\mathcal{S})$, respectively. The profit of $\mathcal{S}$ is the difference:*

$$f(\mathcal{S}) = G(\mathcal{S}) - C(\mathcal{S})$$

In this paper, we measure the crawling cost as $C_{crawl}(\mathcal{S}) = |\mathcal{S}| \cdot f_p + \sum_{W \in \mathcal{W}} f_c \cdot |\mathcal{T}_W|$, which includes a unit cost f_p for training and an extra cost for crawling; de-duplication cost as $C_{de\text{-}dup}(\mathcal{S}) = f_d \cdot | \bigcup_{S \in \mathcal{S}} S|$, which is proportional to the number of facts in the slices; and validation cost as $C_{validate}(\mathcal{S}) = f_v \cdot | \cup_{S \in \mathcal{S}} S \setminus \mathcal{E}|$, which is proportional to the number of new facts in the slices. For our experiments, we use the default values $f_p = 10$, $f_c = 0.001$, $f_d = 0.01$, and $f_v = 0.1$ (we switch to $f_p = 1$ for the running examples in the paper). Appendix A includes more details on the gain and cost functions. MIDAS uses this profit function as the objective function in Definition 8 to identify the set of web source slices that are best-suited for augmenting a given knowledge base.

3. Deriving Web Source Slices

The objective of the slice discovery problem is to identify the collection of web source slices with the maximum total profit. Through a reduction from the set cover problem, we can show that this optimization problem is NP-complete. In addition, because it is a Polynomial Programming problem with a non-linear objective function, the problem is also APX-complete, which means that no constant-factor polynomial approximation algorithm exists.

Theorem 10 (Complexity of slice discovery)**.** *The optimal slice discovery problem is **NP-complete** and it is also **APX-complete** [6].*

In this section, we first present an algorithm, MIDAS$_{alg}$, that solves a simpler problem: identifying the good slices in a single web source (Section 3.1). We then extend the MIDAS$_{alg}$ algorithm to the general form of the slice discovery problem and propose a highly-parallelizable framework, MIDAS, that detects good slices from multiple web sources (Section 3.2).

3.1 Deriving Slices from a Single Source

The problem of identifying high-profit slices *in a single web-source* is in itself challenging. As per Definition 5, given a web source and its extracted facts, any combination of properties, which are derived from the facts, may form a web source slice. Therefore, the number of slices in a single web source can be exponential in the number of extracted facts in the web source. This factor renders most set cover algorithms, as well as existing source selection algorithms [17, 31], inefficient and unsuitable for solving the slice discovery problem since they often need to perform multiple iterations over all slices in a web source. Our approach, MIDAS$_{alg}$, avoids property combinations that fail to match any extracted fact by constructing the slice hierarchy in a bottom-up fashion and guarantees the result quality by further traversing the trimmed slice hierarchy. We demonstrate the two steps algorithm for facts in Example 1 through Figure 7 in Appendix C.

3.1.1 Step 1: Slice hierarchy construction

A key to Midas$_{alg}$'s efficiency is that it constructs slices only as needed, building a slice hierarchy in a bottom-up fashion, and smartly pruning slices during construction. The hierarchy is implied by the properties of slices. For example, slice S_4 (Figure 4) has a subset of the properties of slice S_1, and thus corresponds to a superset of entities compared to S_1. As a result, S_4 is more general and thus an ancestor to S_1 in the slice hierarchy. Midas$_{alg}$ first generates slices at the finest granularity (least general) and then iteratively generates, evaluates, and potentially prunes slices in the coarser levels.

Generating initial slices.

Midas$_{alg}$ creates a set of *initial slices* from the entities in the fact table F_W. Each entity e is associated with the facts $(s, p, o) \in \mathcal{T}_W$ that correspond to that entity $(s = e)$. Each such fact maps to one property (p, o). Thus, the set of all properties that relate to entity e are: $\mathcal{C}_e = \{(p, o) \mid (s, p, o) \in \mathcal{T}_W, s = e\}$.

For each entity e, Midas$_{alg}$ creates one slice for each combination of properties in $\mathcal{C}_e$, such that each property is on a different predicate; if e has a single value for each predicate, there will be a single slice created for e. The algorithm assigns a level to each slice, corresponding to the number of properties that define the slice. These initial slices contain a maximal number of properties and are, thus, canonical slices (Definition 2.1). For example, based on entities in Figure 4, Midas$_{alg}$ creates three slices, S_1, S_2, and S_3, at level 3 from entities e_1, e_3, and e_5, respectively, and one slice, S_4, at level 2 from entities e_2 and e_4.

Bottom-up hierarchy construction and pruning.

Starting with the initial slices, Midas$_{alg}$ constructs and prunes the slice hierarchy in a bottom-up fashion. At each level, Midas$_{alg}$ follows three steps: (1) it constructs the parent slices for each slice in the current level; (2) for each new slice, it evaluates whether it is canonical and prunes it if it is not; (3) if the slice is *canonical*, it evaluates its profit and prunes the slice if the profit is low compared to other available slices. Slices pruned during construction are marked as *invalid*:

(1) Constructing parent slices. At each level, Midas$_{alg}$ constructs the next level of the slice hierarchy by generating the parent slices for each slice in the current level. To generate the parent slices for a slice, Midas$_{alg}$ uses a process similar to that of building the candidate itemset lattice structure in the Apriori algorithm [4]. Given a slice $S = \sigma_C(\mathcal{F}_W)$ with properties $C = \{c_1, ..., c_k\}$, Midas$_{alg}$ generates k parent slices for S, by removing one property from C at a time. For example, Midas$_{alg}$ generates three parent slices for slice S_2: $\{c_2, c_4\}$, $\{c_2, c_6\}$, and $\{c_4, c_6\}$. For each slice we record its children slices; this will be important for removing non-canonical slices safely, as we proceed to discuss.

(2) Pruning non-canonical slices. Midas only reports canonical slices, which are slices with a maximal number of properties (Section 2.1). To identify the canonical slices efficiently, Midas$_{alg}$ relies on the following property.

Proposition 11. *A slice S is canonical if and only if it satisfies one of the following two conditions:*

(1) slice S is an initial slice defined from an entity; or

(2) slice S has at least two children slices that are canonical.

This proposition, proved by contradiction, formalizes a critical insight: the determination of whether a slice is canonical relies on two easily verifiable conditions. For example, in Figure 4, there are two slices, S_4 and S_5, at level 2 and both of them are canonical slices (depicted with solid lines) because 1). S_4 is one of the initial slices, defined by entities e_2 and e_4; and 2). S_5 has two canonical children, S_2 and S_3.

In order to record children slices correctly after pruning, MIDAS$_{alg}$ works at two levels of the hierarchy at a time: it constructs the parent slices at level $l-1$ before pruning slices at level l. The removal of a non-canonical slice S, also updates the children list of the slice's parent, S_p. Each child S_c of the removed slice S becomes a child of S_p if S_c is not already a descendant of S_p through another node. For slices in Figure 4, MIDAS$_{alg}$ prunes the non-canonical slice $(\{c_1, c_3\}, ..., ...)$ and makes its child slice S_1 a direct child of the parent slice $(\{c_3\}, ..., ...)$. However, it does not make S_1 a child of $(\{c_1\}, ..., ...)$ since S_1 is a descendant of $(\{c_1\}, ..., ...)$ through slice node S_4.

(3) Pruning low-profit slices. For the remaining canonical slices, MIDAS$_{alg}$ calculates the statistics to identify and prune slices that may lead to lower profit. This pruning step significantly reduces the number of slices that the traversal (Section 3.1.2) will need to examine. The pruning logic follows a simple heuristic: the ancestors of a slice are likely to be low-profit if the slice's profit is either negative or lower than that of its descendants.

For a slice S, we maintain a set of slices from the subtree of S, denoted by $\mathcal{S}_{LB}(S)$. This set is selected to provide a lower bound of the (maximum) profit that can be achieved by the subtree rooted at S; we denote the corresponding profit as $f_{LB}(S)$. $f_{LB}(S)$ is always non-negative, as the lowest profit, achieved by $\mathcal{S}_{LB}(S) = \emptyset$, is zero. Let $\mathbb{C}_S$ be the set of children of slice S. We compute $f_{LB}(S)$ and update $\mathcal{S}_{LB}(S)$ by comparing the profit of S itself with the profit of the slices in the lower bound sets (S_{LB}) of S's children:

$$f_{LB}(S) = \max\{f(\{S\}), f(\cup_{S_c \in \mathbb{C}_S, f_{LB}(S_c)>0} \mathcal{S}_{LB}(S_c))\}$$

MIDAS$_{alg}$ marks a slice S as low-profit if its current profit is negative or if it is lower than the total profit that can be obtained from the lower bound slices in its subtree ($f_{LB}(S)$). This is because reporting $\mathcal{S}_{LB}(S)$ instead of $\{S\}$ is more likely to lead to a higher profit. For example, among two canonical slices S_4 and S_5 at level 2 in Figure 4, MIDAS$_{alg}$ prunes slice S_4 due to its negative profit. After pruning non-canonical and low-profit slices, MIDAS$_{alg}$ significantly reduces the number of slices at level 2.

Constructing the hierarchy of slices is related to agglomerative clustering [33, 26], which builds the hierarchy of clusters by merging two clusters that are most similar at each iteration. However, MIDAS$_{alg}$ is much more efficient than agglomerative clustering, as we show in our experiments (Section 4).

3.1.2 STEP 2: TOP-DOWN HIERARCHY TRAVERSAL

The hierarchy construction is effective at pruning a large portion of slices in advance, reducing the number of slices we need to consider by several orders of magnitude (Section 4). However, redundancies, or heavily overlapped slices, may still present in the trimmed slice hierarchy, especially for slices that belong to the same subtree. The second step of MIDAS$_{alg}$ traverses the hierarchy top-down to select a final set of slices (Algorithm 1). In this top-down traversal,

Algorithm 1 $\textsc{Midas}_{alg}$: the top-down traversal

```
Require: $\mathcal{E}, F_W, H, L$
    $\mathcal{E}$: existing knowledge base; $F_W$: fact table of the web source $W$; $H$: constructed hierarchy
    $L$: number of levels in the hierarchy; $S.valid$: slice $S$ is not pruned during construction
    $S.covered$: slice $S$ is not covered by the result set $\mathcal{S}$
$\mathcal{S} \leftarrow \emptyset$
for $l$ from 1 to $L$ do
    for $S$ in $H[l]$ do
        if $S.valid$ & $!S.covered$ & $f(\mathcal{S} \cup S) > f(\mathcal{S})$ then
            $\mathcal{S} \leftarrow \mathcal{S} \cup S$
            $S.covered = true$
        if $S.covered$ then
            for $S_c$ in $\mathbb{C}_S$ do
                $S_c.covered = true$
Return $\mathcal{S}$
```

$\textsc{Midas}_{alg}$ prioritizes valid (unpruned) slices at higher levels of the hierarchy, since they are more likely to produce higher profit and cover a larger number of facts than their descendants. We initialize unpruned slices as valid ($S.valid$ =true) but not covered in the result set ($S.covered$ =false).

Proposition 12. $\textsc{Midas}_{alg}$ *has* $O(m^{|\mathcal{P}|})$ *time complexity, where* m *is the maximum number of distinct (subject, predicate) pairs, and* $|\mathcal{P}|$ *is the number of distinct predicates in the web source* W.

According to Theorem 10, the optimal slice discovery problem is APX-complete. Therefore, it is impossible to derive a polynomial time algorithm with constant-factor approximation guarantees for this problem. However, as we demonstrate in our evaluation, $\textsc{Midas}_{alg}$ is efficient and effective at identifying multiple slices for a single web source in practice (Section 4).

3.2 Multiple Slices from Multiple Sources

To detect slices from a large web source corpus, a naïve approach is to *apply* $\textsc{Midas}_{alg}$ *on every web source*. However, this approach leads to low efficiency and low accuracy, as it ignores the hierarchical relationship among web sources from the same web domain, e.g., `http://space.skyrocket.de/doc_sat/apollo-history.htm` is a child of `http://space.skyrocket.de/doc_sat` in the hierarchy. The naïve approach repeats computation on the same set of facts from multiple web sources and returns redundant results. For example, given the facts and web sources in Figure 1, the naïve approach will perform $\textsc{Midas}_{alg}$ on 7 web sources, including 5 web pages, 2 sub-domains, and 1 web domain, and report three slices, "rocket families sponsored by NASA" on web source `http://space.skyrocket.de/doc_lau_fam`, "rocket families sponsored by NASA and started in 1957" on web source `http://space.skyrocket.de/.../atlas.htm`, and "rocket families sponsored by NASA and started in 1971" on web source `http://space.skyrocket.de/.../castor-4.htm`. Even though these three slices achieve the highest profit in their respective web sources, they are as a set redundant and lead to a reduction in the total profit: since the web sources are in the same domain, reporting the latter two slices is redundant and hurts the total profit since the first one already covers all their facts.

In this section, we introduce a highly-parallelizable framework that relies on the natural hierarchy of web sources and explores web source slices in an efficient manner. This framework starts from the finest grained web sources and reuses the derived slices to form the initial slices while processing their parent web source. This framework not only improves the execution efficiency, but also avoids reporting redundant slices over different web sources in the same web domain. Here we highlight three core components in the framework.

Sharding. At each iteration, we take a finer-grained child web source and a list of slices as the input. We generate a one-level-coarser web domain as parent web source (if any) and use it as the key to shard the inputs.

Detecting. After sharding, MIDAS first collects a set of slices for each coarser web source (current) from its finer-grained children, then uses the collected slices to form the initial hierarchy, and applies MIDAS_{alg} to detect slices for the current web source.

Consolidating. To avoid hurting the total profit caused by overlapping slices in the parent and children web sources, MIDAS prunes the slices in the parent web source when there exists a set of slices in the children web sources that cover the same set of facts with higher profit. MIDAS delivers the remaining slices in the parent web source as the input for the next round.

4. Experimental Evaluation

In this section, we present an extensive evaluation of the efficiency and effectiveness of MIDAS over real-world and synthetic datasets. Our experiments show that MIDAS is significantly better than the baseline algorithms at identifying the best sources for knowledge base augmentation. Due to space limit, in this section, we only present experiment results on real-world dataset and we demonstrate the results on synthetic datasets in Appendix D.2.

We ran our evaluation on a ProLiant DL160 G6 server with 16GB RAM, two 2.66GHZ CPUs with 12 cores each, running CentOS release 6.6.

4.1 Datasets

ReVerb/NELL: empty initial KB. We evaluate our algorithms over two real-world datasets. For our experiments on these datasets, we use an empty initial knowledge base and evaluate the precision of returned slices.

ReVerb. The ReVerb ClueWeb extraction dataset [19] samples sentences from the Web using Yahoo's random link service and uses 6 OpenIE extractors to extract facts from these sentences. The dataset includes facts extracted with confidence score above 0.75. Entities and predicates in ReVerb are presented in unlexicalized format; for example, the fact *("Boston", "be a city in", "USA")* is extracted from `https://en.wikipedia.org`. The ReVerb dataset contains 15M facts extracted from 20M URLs.

NELL. The Never-Ending Language Learner project [13] is a system that continuously extracts facts from text in webpages and maintains those with confidence score above 0.75. Unlike ReVerb, NELL is a ClosedIE system and the types of entities follow a pre-defined ontology; for example, in the fact *("concept/athlete/MichaelPhelps", "generalizations", "concept/athlete")*, extracted from *Wikipedia*, the subject "concept/athlete/MichaelPhelps" and object "concept/athlete" are both defined in the ontology. The NELL dataset includes 2.9M facts extracted from 340K URLs.

Evaluation Setup. Due to the scale of the ReVerb and NELL datasets, we report the precision of the returned slices. We manually labeled the correctness of the top-K returned web source slice. Appendix D.1 includes detailed criteria for assigning the labels.

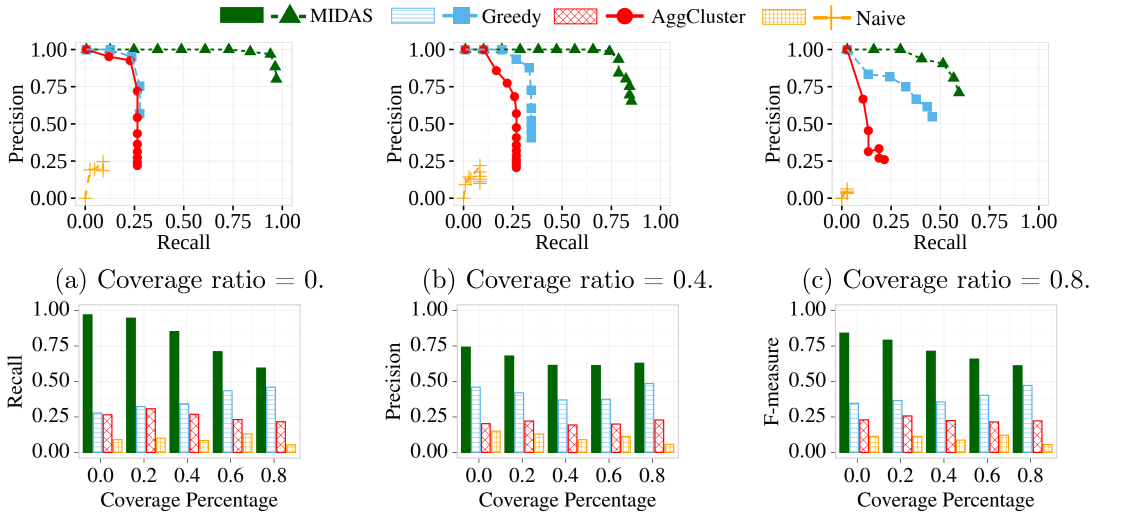
ReVerb-Slim/NELL-Slim: existing KB with adjustable coverage. The ReVerb and NELL datasets provide the input of the slice discovery problem, but they do not contain the optimal output that suggests "what to extract and from which web source". To better evaluate different methods, we generate two smaller datasets, ReVerb-Slim and NELL-Slim, over a 100 sampled web sources in the ReVerb and NELL datasets respectively. We manually label the content of these sources to create an *Initial Silver Standard* of their optimal slices with respect to an empty existing knowledge base. We consider that this optimal, manually-curated set of slices forms a complete knowledge base (100% coverage). We then create knowledge bases of varied coverage, by selecting a subset of the Initial Silver Standard: to create a knowledge base of $x\%$ coverage, we (1) randomly select $x\%$ of the slices from the Initial Silver Standard; (2) build a knowledge base with the facts in the selected slices; (3) use the remaining slices (those not selected in step 1) to form the optimal output for the new knowledge base. The ReVerb-Slim and NELL-Slim datasets contain 859K and 508K facts respectively.
Evaluation Setup. For ReVerb-Slim and NELL-Slim datasets, we select the web sources and manually generate the *Silver Standard.* Appendix D.1 includes more details on the detailed steps for generating the Silver Standard.

4.2 Comparisons

Naïve. There are no baselines that produce web source slices, as this is a novel concept. We compare our techniques with a naïve baseline that selects entire web sources (rather than a slice of their content) based on the number of *new* facts extracted from each source.
Greedy. Our second comparison is a greedy algorithm that focuses on deriving a single slice with the maximum profit from a web source. It relies on our proposed profit function and generates the slice in a web source by iteratively selecting conditions that improve the profit of the slice the most.
AggCluster. We compare our techniques with agglomerative clustering [33], using our proposed objective function as the distance metric. This algorithm initializes a cluster for each individual entity, and it merges two clusters that lead to the highest non-negative profit gain at each iteration. The time complexity of this algorithm is $O(|E|^2 log(|E|)$, where $|E|$ is the number of entities in a web source.
Midas (Section 3.1). Our Midas$_{alg}$ algorithm organizes candidate slices in a hierarchy to derive a set of slices from a single source. Used as the slice detection module in the parallelizable framework of Midas (Section 3.2), it derives slices across multiple sources.
Note that our parallelizable framework in Section 3.2 also supports the alternative algorithms, including Greedy and AggCluster, by adjusting the slice detection algorithm in the *Detecting* phase. Therefore, with the support of our framework, all of these algorithms can easily run in parallel. For all alternative algorithms, we compare their effectiveness based on their *precision*, *recall*, and *f-measure*; and compare their efficiency based on their *total execution time.*

4.3 Evaluation on Real-World Data

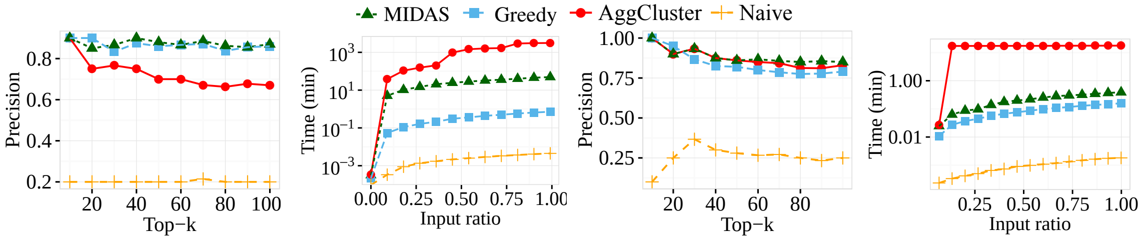
Our evaluation on the real-world datasets includes two components. First, we focus on a smaller version of the datasets, where we can apply our silver standard to better evaluate the result quality using precision, recall, and f-measure across knowledge bases of different cov-



(a) Coverage ratio = 0. (b) Coverage ratio = 0.4. (c) Coverage ratio = 0.8.

(d) Comparison on Recall. (e) Comparison on Precision. (f) Comparison on F-measure.

Figure 5: Comparison of algorithms on the ReVerb-Slim dataset. MIDAS performs better than all alternative algorithms.



(a) Precision (ReVerb). (b) Exe. time (ReVerb). (c) Precision (NELL). (d) Exe. time (NELL).

Figure 6: Top-k precision and execution time on ReVerb and NELL data. The input ratio corresponds to the ratio of sources considered (e.g., a ratio of 0.75 means that 75% of the web sources are considered by each algorithm). MIDAS achieves higher precision and outperforms AGGCLUSTER in terms of efficiency.

erage. Second, we study the performance of all methods on ReVerb and NELL, reporting the precision of the methods' top-k results, for varying values of k, and their execution efficiency.

Slice quality vs. Knowledge Base coverage. For this experiment, we evaluate the four methods on the ReVerb-Slim and NELL-Slim datasets, each with the 100 web sources with labeled silver standard and we run the four methods using input knowledge bases of coverage varying from 0 to 80%. We show the precision-recall curves for three coverage ratios: 0, 0.4, and 0.8 and the precision, recall, and f-measure with increasing coverage ratio from 0 to 0.8. Due to space limit, we only present the result on ReVerb-Slim dataset in Figure 5 and we highlight the major observations of results on the NELL-Slim dataset. The full result can be found in Figure 10 in Appendix D. As shown, MIDAS performs significantly better than the alternative algorithms, especially on the ReVerb-Slim dataset, but there is a noticeable decline in performance with increased coverage. This decline is partially an artifact of our silver standard: since the silver standard was generated against an empty knowledge base, the profit of some of its slices drops as the slices now have increased overlap with existing facts.

Midas tends to favor alternative slices to cover new facts, and may return slices that are not included in the silver standard but are, in fact, better. Greedy performs poorly on both datasets (well under 0.5 for all measures). Its effectiveness is dominated by its recall, which increases with coverage. This is expected since in knowledge bases of higher coverage, there are fewer remaining slices for each source in the silver standard. AggCluster performs poorly for ReVerb-Slim. This is because AggCluster is more likely to make mistakes for datasets with more entities and predicates. In addition, AggCluster requires significantly longer execution time compared to Midas (as demonstrated in Figure 6d). Naïve ranks web sources according to the number of new facts, thus its accuracy heavily relies on the portion of web sources that contain only one high-profit slice. Thus, it achieves similar recall in all different scenarios. Overall, the performance of this baseline is low across the board. Due to the limited size of these two datasets, the execution time of the four methods does not differ significantly. We evaluate the execution efficiency of the methods through our next experiment on the full datasets, ReVerb and NELL.

Precision and efficiency. We further study the quality of the results of all four methods by looking at their top-k returned slices, ordered by their profit, when the algorithms operate on an empty knowledge base. Figures 6a and 6c report the precision for varied values of k up to $k = 100$, for ReVerb and NELL, respectively. We observe that the Naïve baseline performs poorly, with precision below 0.25 and 0.4, respectively. This is expected, as Naïve considers the number of facts that are new in a source, but does not consider possible correlations among them. Thus, Naïve may consider a *forum* or a *news* website, which contains a large number of loosely related extractions, as a good web source slice. In contrast, Midas outperforms Naïve by a large margin, maintaining precision above 0.75 for both datasets. The major disadvantage of Greedy is that it may miss many high-profit slices as it only derives a single slice per web source. However, since we only evaluate the top-100 returns, the precision of Greedy remains high on both datasets. AggCluster performs well on the NELL dataset, but not as well on ReVerb, which includes a higher number of entities and predicates. This is because AggCluster is more likely to reach a local optimum for datasets with more entities and predicates. While AggCluster is comparable to our methods with respect to precision, it does not scale over web sources with larger input, and its running time is an order of magnitude (or more) slower than our methods in most cases. In particular, its efficiency drops significantly on sources with a large number of facts. The NELL dataset contains one source that is disproportionally larger, and dominates the running time of AggCluster (Figure 6d). In ReVerb, most sources have a large number of facts, so the increase is more gradual (Figure 6b). In contrast, the execution time of Greedy, and Midas increases linearly. Naïve is the fastest of the methods, as it simply counts the number of new facts that a web source contributes.

5. Related Work

Knowledge extraction systems extract facts from diverse data sources and generate facts in either fixed ontologies for their subjects/predicate categories, or in unlexicalized format: ClosedIE extraction systems, including KnowledgeVault [15], NELL [13], PROSPERA [28], DeepDive/Elementary [32, 29], and extraction systems in the TAC-KBP competition [14], often generate facts of the first type; whereas OpenIE extraction system [19, 20] normally extract facts of the latter type. In addition, there are many data cleaning and data fusion tools [16, 8] to improve extraction quality of such extraction systems. MIDAS solves the problem of identifying web source slices for augmenting the content of knowledge bases by leveraging on the extracted and cleaned facts. Therefore, the quality of web source slices MIDAS derives significantly relies on the performance of the above systems.

Similar to source selection techniques [17] for data integration tasks, MIDAS also uses customized gain and cost functions to evaluate the profit of a web source slice. However, the slice discovery problem is fundamentally different from source selection problems since the candidate web source slices are unknown.

Collection Selection [11, 10] has been long recognized as an important problem in distributed information retrieval. Given a query and a set of document collections stored in different servers or databases, collection selection techniques retrieve a ranked list of relevant documents: They first perform the selection algorithm on each collection, based on the pre-generated collection descriptions and a similarity metric, and then integrate and consolidate the results into a single coherent ranked list. The slice discovery problem is correlated with the collection selection problem: web sources under the same web domain form a collection, which is further described by the extracted facts; our goal, finding the right web sources for knowledge gaps, can also be considered as a query operate on the collections of web sources. However, instead of a query of keywords, our query is an existing knowledge base. Other than the difference on the queries, there are several additional properties that render these two problems fundamentally different: first, the similarity metrics, which focus on measuring the semantic similarity, in collection selection, do not apply to the slice discovery problem; second, the web sources in a collection in the slice discovery problem form a hierarchy; third, the slice discovery problem not only targets retrieving relevant web sources, but also generating descriptions for the web sources with respect to our query on the fly.

Finally, the slice discovery problem in this paper is related to clustering of entities in a web source [25] . However, it is unclear how to form features for entities. In addition, existing clustering techniques [34], fail to provide any high level description of the content in a cluster, thus they are ill-suited for solving the slice discovery problem.

6. Conclusions

In this paper, we presented MIDAS, an effective and highly-parallelizable system, that leverages extracted facts in web sources, for detecting high-profit web source slices to fill knowledge gaps. In particular, we defined a web source slice as a *selection query* that indicates what to extract and from which web source. We designed an algorithm, MIDAS$_{alg}$, to detect high-quality slices in a web source and we proposed a highly-parallelizable framework to scale MIDAS to million of web sources. We analyzed the performance of our techniques in synthetic data scenarios, and we demonstrated that MIDAS is effective and efficient in real-world settings.

However, there are still many challenges towards solving this problem due to the quality of current extraction systems. There is a substantial number of missing extractions due to the lack of training data and one cannot infer the quality of web sources with respect to such missing extractions. In our future work, we plan to extend our techniques to conquer the limitations of extractions and improve the quality of the derived web source slices.

References

[1] Freebase. https://developers.google.com/freebase.

[2] How google and microsoft taught search to "understand" the web. http://arstechnica.com/information-technology/2012/06/inside-the-architecture-of-googles-knowledge-graph-and-microsofts-satori/. Accessed: 2016-02-05.

[3] Size of wikipedia. https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia.

[4] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, San Francisco, CA, USA, 1994.

[5] G. Angeli, S. Gupta, M. Jose, C. D. Manning, C. Ré, J. Tibshirani, J. Y. Wu, S. Wu, and C. Zhang. Stanford's 2014 slot filling systems. *TAC KBP*, 695, 2014.

[6] M. Bellare and P. Rogaway. The complexity of approximating a nonlinear program. *Mathematical Programming*, 69(1):429–441, 1995.

[7] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *SIGMOD*, pages 11–18, New York, NY, USA, 2004.

[8] J. Bleiholder and F. Naumann. Data fusion. *CSUR*, 41(1):1:1–1:41, 2009.

[9] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, New York, NY, USA, 2008.

[10] J. Callan. Distributed information retrieval. *Advances in information retrieval*, pages 127–150, 2002.

[11] J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems (TOIS)*, 19(2):97–130, 2001.

[12] J. Callan, M. Hoy, C. Yoo, and L. Zhao. Clueweb09 data set. https://www.lemurproject.org/clueweb09.php/, 2009.

[13] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, pages 1306–1313, Atlanta, Georgia, 2010.

[14] H. Chang, M. Abdurrahman, A. Liu, J. T.-Z. Wei, A. Traylor, A. Nagesh, N. Monath, P. Verga, E. Strubell, and A. McCallum. Extracting multilingual relations under limited resources: Tac 2016 cold-start kb construction and slot-filling using compositional universal schema. *Proceedings of TAC*, 2016.

[15] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, pages 601–610, New York, NY, USA, 2014.

[16] X. Dong and F. Naumann. Data fusion: resolving data conflicts for integration. *PVLDB*, 2(2):1654–1655, 2009.

[17] X. Dong, B. Saha, and D. Srivastava. Less is more: Selecting sources wisely for integration. *PVLDB*, 6(2):37–48, 2012.

[18] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and M. Mausam. Open information extraction: The second generation. In *IJCAI*, pages 3–10, Barcelona, Catalonia, Spain, 2011.

[19] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP*, pages 1535–1545, Stroudsburg, PA, USA, 2011.

[20] J. Fan, D. Ferrucci, D. Gondek, and A. Kalyanpur. Prismatic: Inducing knowledge from a large scale lexicalized relation resource. In *Proceedings of the NAACL HLT 2010 first international workshop on formalisms and methodology for learning by reading*, pages 122–127, USA, 2010.

[21] A. L. Gentile and Z. Zhang. Web scale information extraction, ecml/pkdd tutorial, 2013.

[22] L. Getoor and A. Machanavajjhala. Entity resolution: theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.

[23] P. Gulhane, A. Madaan, R. Mehta, J. Ramamirtham, R. Rastogi, S. Satpal, S. H. Sengamedu, A. Tengli, and C. Tiwari. Web-scale information extraction with vertex. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1209–1220, Washington, DC, USA, 2011.

[24] S. Guo, X. Dong, D. Srivastava, and R. Zajac. Record linkage with uniqueness constraints and erroneous values. *PVLDB*, 3(1-2):417–428, 2010.

[25] A. K. Jain and R. C. Dubes. *Algorithms for clustering data.* Upper Saddle River, NJ, USA, 1988.

[26] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. USA, 2009.

[27] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[28] N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 227–236, USA, 2011.

[29] F. Niu, C. Zhang, C. Ré, and J. Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *IJSWIS*, 8(3):42–73, 2012.

[30] R. Pochampally, A. Das Sarma, X. Dong, A. Meliou, and D. Srivastava. Fusing data with correlations. In *SIGMOD*, pages 433–444, New York, NY, USA, 2014.

[31] T. Rekatsinas, X. Dong, and D. Srivastava. Characterizing and selecting fresh data sources. In *SIGMOD*, pages 919–930, New York, NY, USA, 2014.

[32] J. Shin, S. Wu, F. Wang, C. De Sa, C.and Zhang, and C. Ré. Incremental knowledge base construction using deepdive. *PVLDB*, 8(11):1310–1321, 2015.

[33] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1):30–34, 1973.

[34] M. Steinbach, G. Karypis, V. Kumar, et al. A comparison of document clustering techniques. *KDD workshop on text mining*, 400(1):525–526, 2000.

[35] M. Surdeanu. Overview of the tac2013 knowledge base population evaluation: English slot filling and temporal slot filling, 2013.

[36] M. Surdeanu and H. Ji. Overview of the english slot filling track at the tac2014 knowledge base population evaluation. In *TAC*, 2014.

[37] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.

Appendix A. The objective function

The gain of web source slices

The purpose of web source slices is to augment the information in an existing knowledge base. An intuitive way to measure how well a set of slices achieves this objective is to count the new facts that the slices contribute to the knowledge base; this is the *gain* of a set of web source slices.

Definition 13. *Let $\mathcal{E}$ be an existing knowledge base, and $\mathcal{S}$ be a collection of web source slices. The gain of $\mathcal{S}$ with respect to $\mathcal{E}$ is the number of facts selected by these slices that do not appear in $\mathcal{E}$:*

$$G(\mathcal{S}) = \left| \bigcup_{S \in \mathcal{S}} S \setminus \mathcal{E} \right|$$

In our gain function, we take the union of facts and consider the gain of slices in the same web domain as a whole, to avoid double-counting the gain for slices that have overlapping facts. However, we do not penalize the overlap facts across different web domains because acquiring data from multiple web domains helps us evaluate the utility of facts in the same category.

The cost of web source slices

Using web source slices to augment a knowledge base incurs the cost of extracting the corresponding facts. We estimate this cost based on the common knowledge base augmentation procedure [15, 27, 32]. This procedure follows three steps: crawling the web source to extract the facts, de-duplicating facts that already exist in the knowledge base, and validating the correctness of the newly-added facts. Given an existing knowledge base $\mathcal{E}$, a set of web source slices $\mathcal{S}$ from the web sources $\mathcal{W}$, we estimate the cost as follows.

Crawling. The first step of the augmentation process is to crawl and extract the facts in a given web source. This requires training the crawler for the facts in each slice. We use a unit cost f_p to model the cost of training, which includes annotating and schema matching, for each slice. The cost for the rest of the crawling process is proportional to the size of the web source [18]. Measuring the size of web sources is hard due to their diverse design and format; instead, we estimate it based on the total number of facts *extracted from the web sources*, scaled proportional to an adjustable normalization factor f_c:

$$C_{crawl}(\mathcal{S}) = |\mathcal{S}| \cdot f_p + \sum_{W \in \mathcal{W}} f_c \cdot |\mathcal{T}_W|$$

De-duplication. A typical step in the augmentation process is to identify and purge redundant facts before adding them to the knowledge base. This *de-duplication* is often performed through linkage [7, 24, 22] between the facts of the slice and those of the knowledge base. Thus, the de-duplication cost is proportional to the number of facts *selected by the web source slice*, subject to an adjustable normalization factor (f_d):

$$C_{\textit{de-dup}}(\mathcal{S}) = f_d \cdot \left| \bigcup_{S \in \mathcal{S}} S \right|$$

Validation. Before adding facts to a knowledge base, it is essential to verify their validity. The cost of this step is proportional to the *new facts* that the slice contributes, and subject to an adjustable normalization factor (f_v) that depends on the employed validation technique [37, 30]:

$$C_{validate}(\mathcal{S}) = f_v \cdot \left| \bigcup_{S \in \mathcal{S}} S \setminus \mathcal{E} \right|$$

Finally, we compute the cost of slices in the same web domain $C(\mathcal{S})$ as the sum of the respective costs of the crawling, de-duplication, and validation steps.

$$C(\mathcal{S}) = C_{crawl}(\mathcal{S}) + C_{\textit{de-dup}}(\mathcal{S}) + C_{validate}(\mathcal{S}).$$

The four adjustable normalization factors included in the computation of each of the three costs relate to the particular techniques used for the corresponding steps (e.g., different de-duplication methods may result in different values for f_d). In this paper, we set these factors such that they are roughly proportional to the actual execution time of such techniques. However, one can always adjust the setting of these factors. For our experiments, we use the default values $f_p = 10$, $f_c = 0.001$, $f_d = 0.01$, and $f_v = 0.1$ (we switch to $f_p = 1$ for the running examples in the paper). Thus, de-duplication is more costly than crawling, and validation is proportionally the most expensive operation except training.

The objective function

We measure the suitability of a collection of slices $\mathcal{S}$ under the same web domain for augmenting a given knowledge base as the *profit* of the slice, namely, the difference between the gain and the cost.

Definition 14. *Let $\mathcal{S}$ be the web source slices derived from web source W, we denote the gain and the cost of $\mathcal{S}$ with respect to knowledge base $\mathcal{E}$ as $G(\mathcal{S}, \mathcal{E})$ (or $G(\mathcal{S})$) and $C(\mathcal{S}, \mathcal{E})$ (or $C(\mathcal{S})$), respectively.*

$$f(\mathcal{S}) = G(\mathcal{S}) - C(\mathcal{S}).$$

The profit function underlines three important properties for web source slices.

Productivity. MIDAS prioritizes slices that can contribute a larger number of new facts: if S_1 contributes more new facts than S_2, then $G(\{S_1\}) > G(\{S_2\})$.

Specificity. MIDAS prioritizes slices with fewer irrelevant facts: if S_1 on W_1 contributes the same number of new facts as S_2 on W_2, but $|\mathcal{T}_{W_1}| < |\mathcal{T}_{W_2}|$, then $C_{crawl}(\{S_1\}) < C_{crawl}(\{S_2\})$ and $f(\{S_1\}) > f(\{S_2\})$.

Dissimilarity. MIDAS prioritizes slices with fewer facts overlapping with $\mathcal{E}$: if S_1 contributes the same new facts and is extracted from the same web source as S_2, but S_1 has more facts already appearing in $\mathcal{E}$, then $C_{de\text{-}dup}(\{S_1\}) > C_{de\text{-}dup}(\{S_2\})$ and $f(\{S_1\}) < f(\{S_2\})$.

In our objective function $f(\mathcal{S})$, we follow the state-of-the-art procedure and further simplify it with several assumptions: we assume the gain and cost are linear with respect to the number of (new) facts in all slices. However, such assumptions are not inherent in MIDAS; one can adjust the gain and cost functions and use the same methodology to derive high-profit web source slices.

MIDAS uses the above profit function as the objective function $f(\mathcal{S})$ in Definition 8 to identify the set of web source slices that are best-suited for augmenting a given knowledge base. Note that although we define our gain and cost functions as linear functions over the number of (new) facts in all slices, they are non-linear to the input $\mathcal{S}$ since slices in $\mathcal{S}$ may overlap with each other.

Example 15. *In Figure 4, there are three set of slices, $\{S_2, S_3\}$, $\{S_5\}$, and $\{S_6\}$, that cover all the new facts in the web source. Among these slices, reporting S_5 is intuitively the most effective option, since S_5 selects all new facts in the web source and covers zero existing one. We reflect this intuition in our profit function ($f(\mathcal{S})$): slice $\{S_5\}$ has the same gain, but lower de-duplication cost ($6f_d$ vs. $13f_d$), compared to slice $\{S_6\}$ as it contains fewer facts; slice $\{S_5\}$ and slices $\{S_2, S_3\}$ also has the same gain, but $\{S_5\}$ has lower crawling cost (f_p vs. $2f_p$) as it avoids the unit cost for training an additional slice.*

Appendix B. Proofs

B.1 Proof of Theorem 10

Theorem 11. *The optimal slice discovery problem is **NP-complete**.*

Proof. We demonstrate that the slice discovery problem is **NP-complete** by reducing the set cover problem to a the slice discovery problem. Given an instance of a set cover problem with a set of elements, $U = \{u_1, ..., u_m\}$, and a collection of sets, $S = \{S_1, ..., S_n\}$ over the elements such that $\cup_{1 \leq i \leq n} S_i = U$, we construct the following slice discovery problem: for each element $u_i \in U$, we create a fact t_i; for each set $S_i \in S$, we create a slice S'_i such that all the facts that are associated with the elements $u_i \in S_i$ are also covered by slice S'_i; and we set the existing knowledge as empty and adjust the parameters in the profit function with $f_p = \frac{1}{|S|+1}$ and $f_c = f_d = f_v = 0$.

$\Rightarrow$ *The optimal solution of the set cover problem is the optimal solution for the constructed slice discovery problem.*
Let I as the optimal solution for the set cover problem with $|I|$ sets, the corresponding slices J is the optimal solution for the slice discovery problem with profit $|U| - |J|/(|S|+1)$. This is because removing any of the slices in J will hurt the gain by at least 1, but save less than 1 in cost as $\forall k > 0, (|J| - k)/(|S|+1) < 1$. Replacing or adding slices in J will also hurt the gain without improving the cost.

$\Leftarrow$ *The optimal solution for the constructed slice discovery problem is the optimal solution of the set cover problem.*
Let J as the optimal solution for the slice discovery problem, the corresponding sets I is the optimal solution for the set cover problem. First, J must cover all facts in the problem. We may prove this through contradiction: let us assume J does not cover all facts, then any collection of slices, e.g., J', that cover all facts will have a higher profit than J since $|J|/(|S|+1) - |J'|/(|S|+1) < 1$. In addition, among all slice collections that cover all facts, $|J|$ is minimum because otherwise it will not be the optimal solution. As a result, the corresponding collection of sets, I, is also optimal.

Therefore, the slice discovery problem is **NP-Complete**. □

B.2 Proof of Proposition 11 condition (2)

Proposition 12 (2). *A slice S is canonical if slice S has at least two children slices that are canonical.*

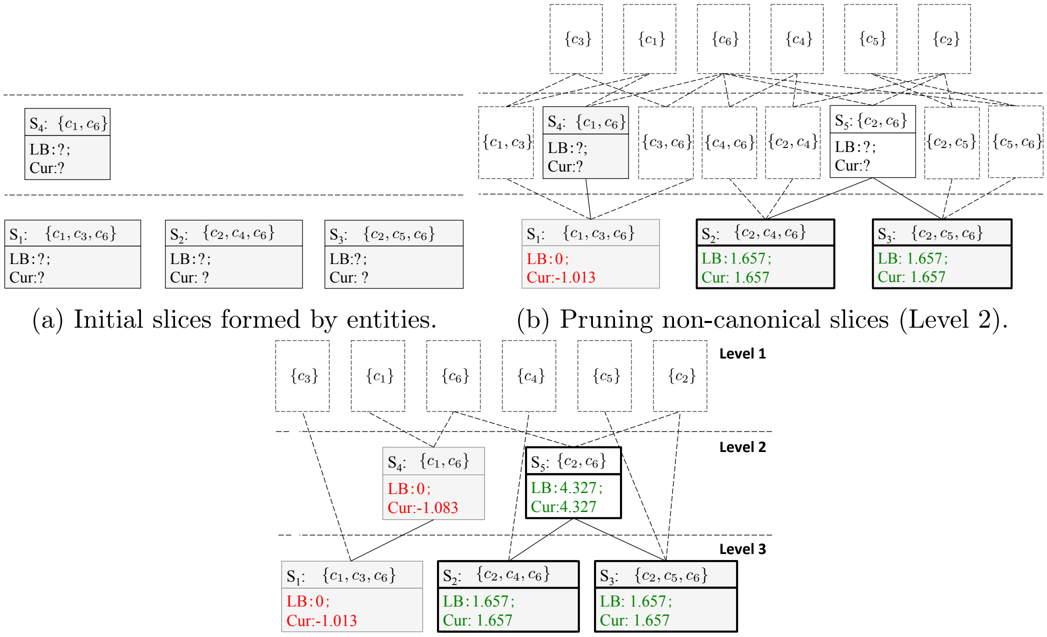
Proof. Let $S_i = (C_i, \Pi_i, \Pi_i^*)$ and $S_j = (C_j, \Pi_j, \Pi_j^*)$ as two children slices of slice $S = (C, \Pi, \Pi^*)$. We say S is also canonical if both S_i and S_j are canonical. We prove this through contradiction: Assume S is not canonical, it means that there must exist another slice $S' = (C', \Pi, \Pi^*)$ such that $C \subset C'$. Since S is the parent of S_i and S_j, we know that $\Pi_i \subset \Pi$, $\Pi_j \subset \Pi$, ${\Pi_i}^* \subset \Pi^*$, and ${\Pi_j}^* \subset \Pi^*$. As S' and S cover the same set of entities and facts, the above conclusion also holds for slice S'. However, since $C \subset C'$, S cannot be the parent of S_i and S_j as there is at least another slice, S', that is between S_i and S (or S_j and S). This contradicts with our initial assumption, therefore S must also be canonical. □

Appendix C. A Running Example

Figure 7 demonstrates the two steps algorithm for identifying slices from a single web source (Section 3.1). During the slice hierarchy construction step, MIDAS$_{alg}$ first creates three slices, S_1, S_2, and S_3, at level 3 from entities e_1, e_3, and e_5, respectively, and one slice, S_4, at level 2 from entities e_2 and e_4.

MIDAS$_{alg}$ then generates parent slices for slices at the lowest level. For example, MIDAS$_{alg}$ generates three parent slices for slice S_2: $\{c_2, c_4\}$, $\{c_2, c_6\}$, and $\{c_4, c_6\}$. While constructing the slice hierarchy, MIDAS$_{alg}$ prunes non-canonical slices. For example, at level 2 in Figure 7b, slices S_4 and S_5 are canonical slices (depicted with solid lines) because S_4 is one of the initial slices, defined by entities e_2 and e_4, and S_5 has two canonical children, S_2 and S_3.

In order to record children slices correctly after pruning, MIDAS$_{alg}$ works at two levels of the hierarchy at a time: it constructs the parent slices at level $l-1$ before pruning slices



(a) Initial slices formed by entities.

(b) Pruning non-canonical slices (Level 2).

(c) Pruning low-profit slices (Level 2).

Figure 7: Constructing the slice hierarchy with MIDAS$_{alg}$ for the facts of Figure 2. LB is short for the profit lower bound ($f_{LB}(S)$), and Cur is short for current profit ($f(S)$). The initial slices, identified by extracted entities, are highlighted in light gray, and identified canonical slices in each step are depicted with solid lines. If the current profit of a slice is lower than the lower bound, we highlight it in red; these slices are low-profit and are eliminated during the pruning stage. The remaining, desired slices are depicted in bold black lines, and have current profit greater or equal to the lower bound.

at level l. For example, in Figure 7, MIDAS$_{alg}$ has constructed the parent slices at level 1, as it is pruning slices at level 2. The removal of a non-canonical slice S, also updates the children list of the slice's parent, S_p. Each child S_c of the removed slice S becomes a child of S_p if S_c is not already a descendant of S_p through another node. In Figures 7b–7c, MIDAS$_{alg}$ prunes the non-canonical slice $(\{c_1, c_3\}, ..., ...)$ and makes its child slice S_1 a direct child of the parent slice $(\{c_3\}, ..., ...)$. However, it does not make S_1 a child of $(\{c_1\}, ..., ...)$ since S_1 is a descendant of $(\{c_1\}, ..., ...)$ through slice node S_4.

Besides non-canonical slices, MIDAS$_{alg}$ also prunes low-profit slices. For example, in Figure 7b there are two canonical slices, S_4 and S_5, remaining at level 2. To prune low-profit slices, MIDAS$_{alg}$ first calculates the statistics of these two slices and then prunes S_4 since its profit is negative. After pruning non-canonical and low-profit slices (Figure 7c), MIDAS$_{alg}$ significantly reduces the number of slices at level 2 from 8 to 1.

Appendix D. Experiment setup and additional results

D.1 Experiment Setup

ReVerb/NELL Datasets Evaluation Setup. Due to the scale of the ReVerb and NELL datasets, we report the precision of the returned slices. We consider a web source slice as

Dataset	# of facts	# of pred.	# of URLs	Existing KB
ReVerb	15M	327K	20M	Empty
NELL	2.9M	330	340K	Empty
ReVerb-Slim	859K	33K	100	Adjustable
NELL-Slim	508K	280	100	Adjustable

Figure 8: Statistics of real-world datasets.

URL	Desired slices description
`http://www.nationsencyclopedia.com`	Information about nations
`https://www.drugs.com`	Medicinal chemical
`https://www.citytowninfo.com/places`	US city profiles
`http://www.u-s-history.com/`	Events in US history
`http://blogs.abcnews.com`	No desired slice
`http://voices.washingtonpost.com`	No desired slice

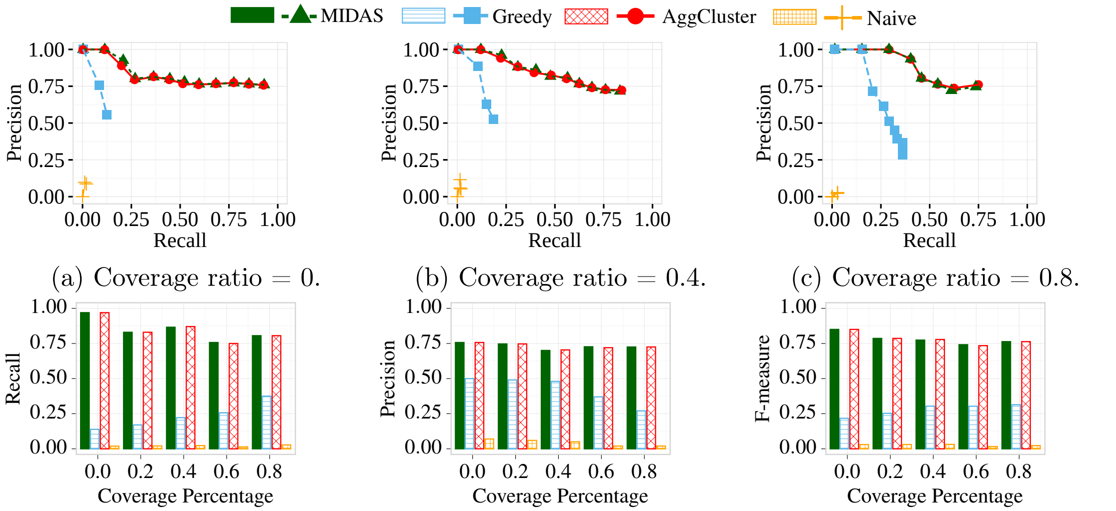
Figure 9: A snapshot of selected web sources in the silver standard: Among 100 selected web sources, 50 of them contain at least one high-profit slice.

"correct" if it satisfies two criteria: (1), whether it provides information that is absent from the existing knowledge base; and (2), whether it allows for easy annotation. We implement these two criteria based on two statistics: (a) The ratio (R_{new}) of new facts for the covered entities; (b) The ratio (R_{anno}) of entities that provide homogeneous information. To evaluate a given web source slice, we first randomly select K or fewer entities and their web pages; then, we display them to human workers, together with the slice description and existing facts associated with the entity; finally, we ask human workers to label the above two statistics. For this set of experiments on ReVerb and NELL, since the initial knowledge base is empty, the first ratio R_{new} becomes binary: it equals to 1.0 when there exist facts of the associated entities, or 0.0 otherwise. In our experiment, we set $K = 20$ and mark a slice as "correct" if both statistics are above 0.5.

ReVerb-Slim/NELL-Slim Datasets Evaluation Setup. For ReVerb-Slim and NELL-Slim datasets, we select the web sources and generate the *Initial Silver Standard* as follows: (1) we manually select 100 web sources, such that 50 of them contain at least one high-profit slice, with respect to an empty knowledge base; (2) we apply all algorithms on the selected web sources with an empty knowledge base; (3) we manually label slices and web sources returned by the algorithms, and add those labeled as correct to the Initial Silver Standard. We demonstrate a snapshot of the selected web sources and the description of the labeled silver standard slices for the ReVerb-Slim dataset in Figure 9. As described earlier, the initial silver standard allows us to adjust the coverage of the existing knowledge base and the optimal output. In our experiment, we evaluate the performance of the different methods against knowledge bases of varied coverage, ranging from 0% (empty KB) to 80%.

D.2 Evaluation on Synthetic Data

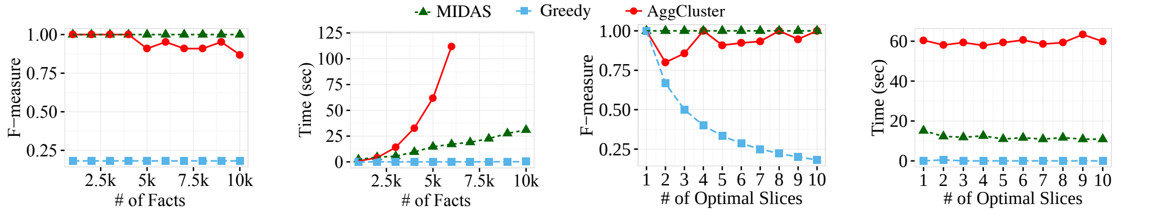
We use synthetic data to perform a deeper analysis of the tradeoffs between the three algorithms, Greedy, Midas, and AggCluster, that use our objective function and to study the effectiveness of the pruning strategies of our proposed algorithm, Midas. We



(a) Coverage ratio = 0. (b) Coverage ratio = 0.4. (c) Coverage ratio = 0.8.

(d) Comparison on Recall. (e) Comparison on Precision. (f) Comparison on F-measure.

Figure 10: Comparison of algorithms on the NELL-Slim dataset. Greedy and Naïve perform poorly. AggCluster competes with Midas, but is significantly slower (Figure 6d).



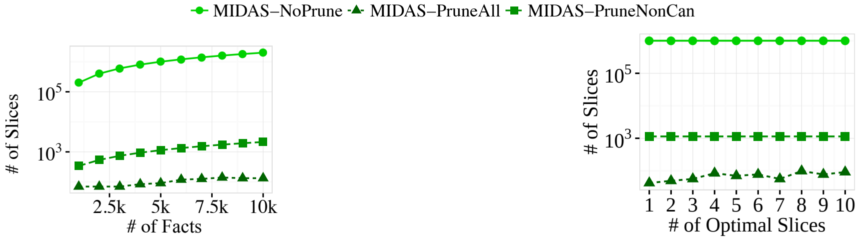
(a) Comparison on accuracy. (b) Comparison on runtime. (c) Comparison on accuracy. (d) Comparison on runtime.

Figure 11: Comparison of the methods that use our objective function. Midas outperforms AggCluster in effectiveness and efficiency. Greedy is less effective than Midas, but it is faster.

create synthetic data by randomly generating facts in a web source based on user-specified parameters: the number of slices k, the number of optimal slices $m \leq k$ (output size), and the number of facts n (input size): For each slice, we first generate its selection rule that consists 5 conditions and then creates $n \cdot 1\%$ entities in this slice. To better simulate the real-world scenario, we also introduce some randomness while generating the facts in the optimal slice: for each entity, the probability of having a condition in the corresponding selection rule is above 0.95 and the probability of having a condition absent from the selection rule is below 0.05. Among k slices, we select m of them as optimal slices and construct the existing knowledge base accordingly: for non-optimal slices, we randomly select 0.95 of their facts and add them in the existing knowledge base. In addition, we ensure that each optimal web source slice covers at least 5% of the total input facts.

D.2.1 Comparison on accuracy and efficiency

We compare the Greedy, Midas, and AggCluster in terms of their total running times



(a) Vary # of facts (n).

(b) Vary # of optimal slices (m).

Figure 12: Midas's pruning strategies are effective at reducing the hierarchy size by several orders of magnitude.

and their f-measure scores (Figure 11). In our first experiment, we fix $b = 20, m = 10$ (10 optimal slices out of 20 slices in a web source), and range the number of facts from 1,000 to 10,000. Midas remains highly accurate in detecting web source slices in all these settings. However, due to its time complexity, the execution time of Midas grows linearly with the number of facts. AggCluster tends to make more mistakes when there are more facts and its execution time grows at a significantly higher rate than Midas. The greedy algorithm, Greedy, runs much faster than the other algorithms, but it can only detect one out of ten optimal slices.

In our second experiment, we use a web source with 5000 facts ($n = 5000$) on 20 slices ($b = 20$), and vary the number of optimal slices in the web source from 1 to 10. We report the execution time and f-measure in Figures 11d and 11c, respectively. AggCluster is much slower than Midas and it fails to identify the optimal slices under several settings. This is expected as AggCluster only combines two slices at a time, thus it needs more iterations to finish and the probability of reaching a local optimum is much higher than Midas. Notably, Midas achieves perfect f-measure across the board. Greedy is three times faster than Midas, but its f-measure score declines quickly as the number of slices increases. This is expected, as Greedy can only retrieve a single high-profit slice. At the same time, Greedy is able to find the optimal slice when there is only one.

D.2.2 Evaluating the pruning strategy of MIDAS

Midas prunes non-canonical slices and low-profit slices while constructing the hierarchy (Section 3.1.1). Here, we further study the effectiveness of these two pruning strategies by comparing the number of slices in the constructed hierarchy. More specifically, using synthetic data, we compare Midas with both non-canonical and low-profit slice pruning (Midas-PruneAll), Midas with the pruning of non-canonical slices strategy only (Midas-PruneNonCan), and Midas with no pruning strategy (Midas-NoPrune). Figure 12a shows the number of slices with increasing number of facts ($n = 1000 \sim 10000$) and a fixed number of optimal slices ($m = 10$). Midas-PruneAll generates significantly fewer slices than Midas-PruneNonCan. Midas-NoPrune needs to examine every non-empty slice in the web source, thus produces several orders of magnitude more slices than Midas-PruneAll and Midas-PruneNonCan. Figure 12b demonstrates the number of slices with fixed number of facts ($n = 5000$) and an increasing number of optimal slices ($m = 1 \sim 10$). Similar to our observation in the previous experiment, MIDAS-PRUNENONCAN and MIDAS-NOPRUNE generate significantly more slices than MIDAS-PRUNEALL across all settings.