Chapter 6

RuDiK: Affordable Rules Discovery in Knowledge Bases

So far, the focus of this thesis has targeted classic web data extraction: we extract entities from the Web, along with their attributes, in order to store them in some structured format. A novel research direction is recently focusing on extracting data from the Web to construct knowledge bases (KBs), in order to structurally store entities and relationships among them. Given the different nature of target data, KBs are usually not stored in relational databases. In this chapter we move the attention from classic web data extraction to KBs built from Web sources. More specifically, we consider RDF KBs, the most popular format where data is stored as RDF triples.

Building large RDF KBs has become a popular trend in information extraction. Differently from classic relational databases, KBs store information in the form of triples, where a *predicate* expresses a binary relation[1] between a *subject* and a *object*. KB triples, often referred as facts, store information about real-world entities, such as "Scott Eastwood is the child of Clint Eastwood", "Michelle Obama is the spouse of Barack Obama", or "Larry Page is the founder of Google". The most popular research efforts in this direction can be currently grouped into two strands, according to the creation technique they rely on: *human crafted* KBs (DBPedia [29], FreeBase [31], Wikidata [137]), and automated *information extraction* (DeepDive [128], Yago [133], NELL [34], TextRunner [21]). Recent advances in KBs have also attracted interests from the industrial world, as evident from several companies directions (Facebook Graph[2],

[1]We will use the two terms "predicate" and "relation" interchangeably in this chapter
[2]`https://developers.facebook.com/docs/graph-api`

Google Knowledge Vault [57], Wallmart [56]).

KBs are built by extracting information from Web sources. Given the abundance of Web data, KBs nowadays end up being prohibitively large – the online version of WIKIDATA contains approximately more than 1 billion facts and 300 million different entities[3]. Furthermore, usually KBs do not expose a schema that clearly defines instance data. The set of predicates is unknown a-priori, and adding new predicates/facts is just a matter of inserting new triples in the KB without any schema/integrity check. As a direct consequence, the amount of incompleteness and inconsistencies in KBs is significantly larger than classic databases [134]. For example, the standard *closed world assumption* (CWA) does not longer hold in KBs [57, 78]. We cannot assume that a fact that is not stated in a KB is false, but rather we should label it as *unknown*. This is known as *open world assumption* (OWA).

Despite being erroneous and incomplete, KBs carry such a big amount of information that they can be inspected to find additional information or constraints over existing data. In this chapter we investigate the problem of discovering *declarative rules* over RDF KBs. More specifically, we target the discovery of first-order Horn Rules constrained by some language biases. We define two different types of rules: *(i)* positive rules, used to enrich the KB with new facts and thus make it more complete; *(ii)* negative rules, used to spot logical inconsistencies and repair erroneous triples.

Example 17. As an example, consider a KB that contains information about parent and child relationships. A potential positive rule could be:

$$\texttt{parent}(b, a) \Rightarrow \texttt{child}(a, b)$$

which states that if a is parent of b, then b is child of a. If the KB states that "Clint Eastwood is the parent of Scott Eastwood", we can infer the fact that "Scott Eastwood is the child of Clint Eastwood". On the other hand, a negative rule may state that:

$$\texttt{birthDate}(a, v_0) \land \texttt{birthDate}(b, v_1) \land v_0 > v_1 \Rightarrow \neg\texttt{child}(a, b)$$

[3]`https://query.wikidata.org/`

The above rule expresses the concept that b cannot be child of a if a was born after b. By instantiating the above rule on the KB, we may discover erroneous triples where a child is born before her parent.

Other than enriching and cleaning KBs, discovering declarative rules brings multiple benefits. We can discover negative facts (absent in KBs), bringing the KB world a step closer to the classic database CWA. We can help domain experts in automatically defining rules to maintain and curate Big Data industrial systems [81]. We can enhance Machine Learning tasks by providing meaningful training examples [122, 128].

Discovering declarative rules from data sources is a long standing research challenge. Inductive Logic Programming (ILP) was the first discipline to mine Horn Rules from background knowledge and training examples [114]. Similarly, a significant body of work has addressed the problem of discovering constraints over relational databases. The most popular constraint families are Functional Dependencies [17, 91, 145] and Denial Constraints [39]. However, current KBs pose new challenges that make these approaches unsuitable. First, the size of modern KBs. Classic ILP systems cannot cope with large KBs and adopting database techniques would require the materialisation of all possible predicates combinations into relational tables. Second, the OWA that rules KBs. KBs contain only positive statements, and since we cannot rely on the CWA, we cannot derive negative statements as counter examples – most of classic databases approaches rely on the presence of positive and negative examples or on the CWA. Third, ILP and database techniques usually make the assumption that data is either clean or has a negligible amount of errors. We will show that KBs present a high percentage of errors and we need techniques that are noise tolerant.

Novelty. In this chapter we present RuDiK (Rules Discvoery in Knowledge Bases), a novel system for the discovery of positive and negative Horn Rules over KBs. To the best of our knowledge, RuDiK is the first system capable of discovering both positive and negative rules using the same algorithm. Moreover, classic KBs rules discovery systems assume that the KB has to fit in memory to have acceptable performance [15, 78] (TODO: cite Sigmod 2 papers). We drop such an assumption and propose a disk-based solution that loads into memory only a small portion of the KB. This is a key point not only because we can discover rules on any size KB, but also because we can enlarge the expressive power of the considered rules to include smart comparisons among constants and numbers, including inequalities.

Contribution 1. We first formally define the problem of rules discovery over erroneous and incomplete KBs. The input of the problem consists of a target predicate, uniquely identified by a set of positive and negative examples. The ideal solution of the problem is a set of rules that cover all positive examples, and as few negative examples as possible. The problem can be naturally mapped to the well-known weighted set cover problem, which is proven to be NP-complete.

In contrast with traditional ranking of rules based on a measure of support [55, 78, 113, 126], our novel problem definition inspired by set cover allows the identification of a subset of useful rules to be exposed to the user. The main benefit of such an approach is to relieve the user from the definition of a suitable threshold that splits good and bad rules. We will show that properly setting such thresholds is not trivial (see Section 6.4.2), and we incorporate this aspect in the problem definition.

Contribution 2. We give a `log(`k`)` approximation of the problem close in spirit to the greedy algorithm for set cover. k is the maximum number of positive examples covered by a single rule. The algorithm we propose in RuDiK alternates and combines two different phases: *(i)* it materialises the KB as a directed graph and discovers rules by navigating valid paths on the graph; *(ii)* at each iteration it follows the most promising path in a A^* traversal fashion, allowing the pruning of unpromising paths.

Previous approaches for rules discovery assume that the KB has to fit in main memory to deliver acceptable performance [15, 78] (TODO: cite 2 sigmod papers). Unfortunately, KBs nowadays have sizes that can easily exceed hundreds of GBs and previous works require powerful resources to handle them. With our rules generation based on graph traversal we materialise only a small portion of the KB that is needed for the discovery, generating nodes and edges *on demand* by querying the disk whenever the navigation of a specific path is required. Moreover, the A^* graph traversal allows the pruning of unpromising paths that are therefore never materialised. Thus we significantly reduce the search space, cutting down the running time of ten times in best case scenario.

Contribution 3. We experimentally verify the performance of rules discovery in RuDiK using both real-world and synthetic datasets (Section 6.4). The evaluation is carried on the 3 most popular and widely used KBs, namely DBPEDIA [29], YAGO [132], and WIKIDATA [137]. We show that our algorithm is general enough to deliver accurate rules both in the positive and negative setting, and it clearly outperforms state-of-the-art systems both in final accuracy and running time. Eventually we give an example of application of our rules discovery approach, providing representative examples for Machine Learning algorithms (Section 6.4.3).

Roadmap. The rest of the chapter is structured as follows. Section 6.1 introduces basic preliminaries and the discovery problem tackled by RuDiK. In Section 6.2 we present a graph-based solution to generate the universe of all possible rules, while Section 6.3 is the main contribution of the chapter: we outline an approximate solution of the problem along with some optimisation techniques. Section 6.4 empirically evaluates RuDiK, and the most relevant works are surveyed in Section 6.5.

6.1 Preliminaries and Definitions

The work of this thesis focuses on discovering rules from RDF[4] Knowledge Bases (KBs). An RDF KB is a database that represents common information through RDF triples $\langle s, p, o \rangle$, where a *subject* is connected to a *object* via a *predicate*. Triples are often called *facts*. As an example, the fact that Scott Eastwood is the child of Clint Eastwood could be represented with the triple $\langle Clint\_Eastwood, child, Scott\_Eastwood \rangle$. RDF KB triples respect the following constraints:

- triple subjects are always *entities*, i.e., concepts from the real world;
- triple objects can be either entities or *literals*. Literals represent primitive types – numbers, dates, and strings;
- triple predicates specifies real-world relationships between subjects and objects.

[4]`https://www.w3.org/RDF/`

If we compare the KB model with object oriented programming languages, entities correspond to complex objects with properties and attributes, while literals correspond to primitive types such as integers, chars and strings.

Differently from, e.g., relational databases, KBs do not have a schema that instance data is forced to follow. The set of predicates is unknown a-priori, and adding new predicates is just a matter of inserting new triples having those predicates. Sometimes KBs present *T-Box* facts in order to define classes, domain/co-domain types for predicates, and relationships among classes. While the T-Box can be useful in order to check integrity and consistency within a KB, very often such information is incomplete or not present at all. Hence our focus in this thesis is only on the *A-Box*, the set of facts that describe instance data. If on the one hand this model allows great flexibility, on the other hand the likelihood of introducing errors and inconsistencies is much higher than traditional schema-guided databases.

6.1.1 Language

Inspired by Inductive Logic Programming [114], the goal of our work is to discover first-order logical formulas in KBs. More specifically, we target the discovery of *Horn Rules*. A Horn Rule is a disjunction of *atoms* with at most one unnegated atom. When written in the implication form, Horn Rules have one of the following format:

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow B \qquad\qquad A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow \neg B$$

where $A_1 \wedge A_2 \wedge \cdots \wedge A_n$ consists of the *body* of the rule (a conjunction of atoms), while B is the *head* of the rule (a single atom). The head of the rule is either unnegated (left) or negated (right). We call the former *definite clause* or more simply *positive rule*, while the latter *goal clause* or *negative rule*. In the context of KBs, an atom is a predicate connecting or two variables, or two entities, or an entity and a variable. For simplicity, we represent an atom with the notation $\texttt{rel}(a, b)$, where $\texttt{rel}$ is the name of the predicate, and a and b are either variables or entities. In the context of Horn Rules, all variables appearing in a rule are implicitly universally quantified. Given a Horn Rule r, we define r_{body} as the body of the rule, and r_{head} as the head of the rule. We define the variables appearing in the head of the rule as the *target variables*. The first rule of Example 17 shows a plausible positive rule, while the second rule is a potential negative rule. a and b are the target variables in both cases.

Furthermore we extend the language in order to include *literals comparison*. A literal comparison is a special atom $\texttt{rel}(a, b)$, where $\texttt{rel} \in \{<, \leq, \neq, >, \geq\}$, and a and b can only be assigned to literal values except if $\texttt{rel}$ is equal to $\neq$. In such case a and b can be also assigned to entities. We will explain later on in the thesis why this exception is important.

Given a KB $\mathcal{K}$ and an atom $A = \texttt{rel}(a, b)$ where a and b are two entities, we say that A holds over $\mathcal{K}$ iff $\langle a, \texttt{rel}, b\rangle \in \mathcal{K}$. Given a KB $\mathcal{K}$ and an atom $A = \texttt{rel}(a, b)$ with at least one variable, we say that A can be instantiated over $\mathcal{K}$ if there exists at least one entity from $\mathcal{K}$ for each variable in A such that A holds over $\mathcal{K}$. Transitively, given a body of a rule r_{body} and a KB $\mathcal{K}$, we say that r_{body} can be instantiated over $\mathcal{K}$ if every atom in r_{body} can be instantiated.

Eventually, we introduce two language biases in order to avoid the explosion of the search space. Following the biases introduced by other approaches for rules discovery in KBs [78] (TODO: cite Sigmod Paper Ontological Path...), we introduce such restrictions in order to speed up the discovery process. Language biases introduce a compromise between the expressiveness of target rules and the complexity of the discovery problem. Therefore we define a rule *valid* iff it follows the following constraints.

Connectivity. We say that in a rule an atom A_1 can be reached by an atom A_2 iff A_1 and A_2 share at least one variable or one entity. The connectivity constraint requires that every atom in a rule must be *transitively* reached by any other atom in the rule.

Repetition. Every variable in a rule must appear at least twice. Since target variables already appear once in the head of the rule, the repetition constraint limited to the body of a rule requires that each variable that is not a target variable must appear at lest twice, while every target variable must appear at least once.

Language restrictions obviously limit the output of the discovery algorithm to a subset of all plausible rules. We will show in Section 6.2 how these restrictions can significantly speed up the discovery process through a fast disk-based approach that leverages on these limitations.

6.1.2 Rules Coverage

Given a pair of entities (x, y) from a KB $\mathcal{K}$ and a Horn Rule r, we say that r_{body} *covers* (x, y) if $(x, y) \models r_{body}$. In other words, given a Horn Rule $r = r_{body} \Rightarrow \texttt{r}(a, b)$, r_{body} covers a pair of entities $(x, y) \in \mathcal{K}$ iff we can substitute a with x, b with y, and the rest of the body can be instantiated over $\mathcal{K}$. Given a set of pair of entities $E = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ and a rule r, we denote by $C_r(E)$ the *coverage* of r_{body} over E as the set of elements in E covered by r_{body}: $C_r(E) = \{(x, y) \in E | (x, y) \models r_{body}\}$.

Given the body r_{body} of a Horn Rule r, we denote by r^*_{body} the *unbounded body* of r. The unbounded body of a rule is obtained by substituting each atom in r that contains a target variable with a new atom where the target variable is connected to a new unique variable. As an example, given $r_{body} = \texttt{rel}_1(a, v_0) \wedge \texttt{rel}_2(v_0, b)$ where a and b are the target variables, $r^*_{body} = \texttt{rel}_1(a, v_1) \wedge \texttt{rel}_2(v_2, b)$. While in r_{body} the target variables are bounded to be connected to the same variable v_0, in r^*_{body} the target variables are not bounded to share the same variable. Given a set of pair of entities $E = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ and a rule r, we denote by $U_r(E)$ the *unbounded coverage* of r^*_{body} over E as the set of elements in E covered by r^*_{body}: $U_r(E) = \{(x, y) \in E | (x, y) \models r^*_{body}\}$. Note that, given a set E, $C_r(E) \subseteq U_r(E)$. In other words, the unbounded coverage of a rule over E always contains all the elements of the coverage over E.

Example 18. Given the negative rule r of Example 17 and a KB $\mathcal{K}$, we denote with E the set of all possible pairs of entities in $\mathcal{K}$. The coverage of r over E $(C_r(E))$ is the set of all pairs of entities $(x, y) \in \mathcal{K}$ such that both x and y have the `birthDate` information and x is born after y, while the unbounded coverage of r over E $(U_r(E))$ is the set of all pairs of entities (x, y) such that both x and y have the `birthDate` information, no matter what the relation between the two birth dates is.

The unbounded coverage is essential to distinguish between missing and inconsistent information: if for a pair of entities (x, y) the `birthDate` information is missing for either x or y (or both), we cannot say whether x was born before or after y, therefore we cannot state whether the negative rule of Example 17 covers (x, y) or not. On the other hand, if both x and y have the `birthDate` information and x is born before y, we can affirm that the negative rule of Example 17 does not cover (x, y). Given that modern KBs are largely incomplete [57, 112], discriminating between missing and conflicting information becomes of paramount importance.

We can now define the coverage and the unbounded coverage for a set of rules $R = \{r_1, r_2, \cdots, r_n\}$ as the union of individual coverages:

$$C_R(E) = \bigcup_{r \in R} C_r(E) \qquad U_R(E) = \bigcup_{r \in R} U_r(E)$$

Our problem tackles the discovery of positive (negative) rules for an input given predicate (target predicate). We uniquely identify a predicate with two different sets of pairs of entities. G – *generation set*: G contains good examples for the target predicate (G contains examples of parents and children if we are discovering positive rules for a child predicate). V – *validation set*: V contains counter examples for the target predicate (pairs of people that are not in a child relation). We will explain in Section 6.2.1 how to generate these two sets for a given predicate. Note that our approach is not less generic than those for mining rules for an entire KB (e.g., [15, 78]): it is true that we require a target predicate as input, however we can generically apply our setting for every predicate in the KB and compute rules for each of them (see Section 6.4.2).

We can now formalise the *exact discovery problem*.

Definition 6. Given a KB $\mathcal{K}$, a set of pairs of entities G from $\mathcal{K}$, a set of pairs of entities V from $\mathcal{K}$ such that $G \cap V = \emptyset$, and a universe of Horn Rules R, a solution for the *exact discovery problem* is a subset R' of R such that:

$$R_{opt} = \underset{|R'|}{\operatorname{argmin}}(R' | (C_{R'}(G) = G) \wedge (C_{R'}(V) \cap V = \emptyset))$$

The ideal solution is a set of rules that covers all examples in G, and none of the examples in V. Note that given a pair of entities (x, y), we can always generate a Horn Rule whose body covers only (x, y) by assigning target variables to x and y.

Unfortunately, since the solution is not allowed to cover any element in V, in the worst case the exact solution may be a set of rules such that each rule covers only one example in G, making such set of rules difficult to use and understand.

6.1.3 Weight Function

In order to allow flexibility and errors in both G and V, we drop the strict requirement of not covering any element of V. However, since covering elements in V is an indication of potential errors, we want to limit the coverage over V to the minimum possible. We therefore define a *weight* to be associated with a rule.

Definition 7. Given a KB $\mathcal{K}$, two sets of pair of entities G and V from $\mathcal{K}$ such that $G \cap V = \emptyset$, and a Horn Rule r, the weight of r is defined as follow:

$$w(r) = \alpha \cdot (1 - \frac{\mid C_r(G) \mid}{\mid G \mid}) + \beta \cdot (\frac{\mid C_r(V) \mid}{\mid U_r(V) \mid}) \tag{6.1}$$

with $\alpha, \beta \in [0, 1]$ and $\alpha + \beta = 1$.

The weight is a value between 0 and 1 that captures the *goodness* of a rule w.r.t. G and V: the better the rule, the lower the weight – perfect rules would have a weight of 0. The weight is made of two components normalised by the two parameters α and β. *(i)* The first component captures the coverage over the generation set G – the ratio between the coverage of r over G and G itself. Note that if r covers all elements in G, then this component is 0 because of the subtraction from 1. *(ii)* The second component aims at quantifying potential errors of r, or rather the coverage over V. The coverage over V is not divided by total elements in V, because for those elements in V that do not have predicates stated in r_{body} we cannot be sure whether such elements are not covered by r. Thus we divide the coverage over V by the unbounded coverage of r over V. Ideally this number is close to 0. The parameters α and β are used to give relevance to each component. We would set a high β if we want to discover rules with high precision that identify few mistakes, or we would set a high α if we are more interested in recall and the discovered rules should identify as many examples as possible.

Example 19. W.r.t. the negative rule r of Example 17, given two sets of pair of entities G and V from a KB $\mathcal{K}$, the two components of w_r are computed as follow: *(i)* the first component is computed as 1 minus number of pairs (x, y) in G where x is born after y divided by the total number of elements in G; *(ii)* the second component is the ratio between number of pairs (x, y) in V where x is born after y and number of pairs (x, y) in V where the date of birth (for both x and y) is available in $\mathcal{K}$.

Definition 8. Given a set of rules R, the weight for R is defined as:

$$w(R) = \alpha \cdot (1 - \frac{\mid C_R(G) \mid}{\mid G \mid}) + \beta \cdot (\frac{\mid C_R(V) \mid}{\mid U_R(V) \mid})$$

Assigning a weight to one or multiple rules allows us to take into consideration an important aspect of modern KBs: the presence of errors. We will show in the experimental evaluation that very often universally correct rules have a significant coverage over V, which corresponds to errors in the KB. The exact discovery problem implies the absence of errors in the input KB, however such an assumption is too strong for modern KBs that are usually automatically built from external web sources [29, 57, 132] and thus inconsistencies are common [134].

6.1.4 Problem Definition

We can now state the approximate version of the problem.

Definition 9. Given a KB $\mathcal{K}$, two sets of pair of entities G and V from $\mathcal{K}$ where $G \cap V = \emptyset$, a universe of rules R, and a w weight function for R, a solution for the *approximate discovery problem* is a subset R' of R such that:

$$R_{opt} = \underset{w(R')}{\operatorname{argmin}}(R' | R'(G) = G)$$

We can map this problem to the well-known weighted set cover problem, which is proven to be a NP-complete problem [43]. The universe corresponds to G and the input sets are all the possible rules defined in R.

Since we want to minimise the total weight of the output rules, the approximate version of the discovery problem aims to cover all elements in G, and as few as possible elements in V. Since for each element $g \in G$ there always exists a rule that covers exactly only g (single-instance rule), an optimal output is always guaranteed to exist. We expect such output to be made of some rules that cover multiple examples in G, while remaining examples in G to be covered by single-instance rules. In the best-case scenario a single rule covers all elements in G and none of the elements in V.

Section 6.3 will describe a greedy polynomial algorithm to find a good solution for the approximate discovery problem.

6.2 Rules Generation

We introduce RuDiK, a disk-based approach to discover first-order Horn Rules in RDF KBs. The first task RuDiK needs to address is the generation of the universe of all possible rules R. Each rule in R must cover one or more examples of the generation set G.

In this setting, the universe of all possible rules can be generated by just inspecting elements of G. In fact, a rule that does not cover any element of G will never be part of the optimal solution, since it will not give any contribution to the set cover problem. This is a key part of our generation approach: we do not inspect the entire KB, but just entities belonging to G. The smaller the size of G is, the smaller is the search space for rules generation. On the contrary, recent KB rule mining approaches tackle the problem by searching for valid rules over the entirety of the KB. This is typically done by initially looking at all facts that share a common predicate, and then expanding these facts with other predicates that share common entities [78] (TODO : CITE Ontological path finding). Alternatively other approaches leverage on well established relational database techniques such as functional dependencies discovery

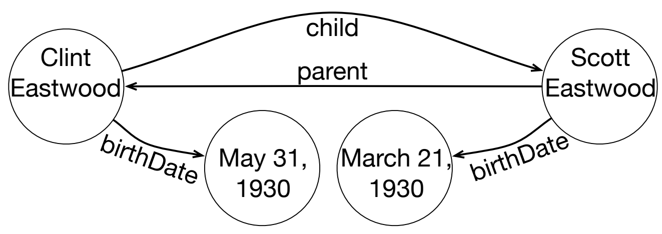


Figure 6.1: Graph Portion of DBPedia

(TODO: cite SIGMOD DEMO Data Lakes).

The generation problem is simple but presents significant scalability issues. The number of all possible rules increases exponentially with the size of the KB and enumerating all of them means exploring a huge search space. This is the main reason why previous approaches solve the problem by loading the entire KB into main memory and by aggresively indexing triples on subject, object, and predicate. Unfortunately, modern KBs can easily exceed the size of hundreds of GBs, making a memory-based solution unfeasible on common machines. We introduce in RuDiK a novel rules generation technique that inspects only a portion of the KB. On the one hand, we load into main memory only a small fraction of the database. On the other hand, this model allows the extension of the language to be more expressive than previous works. One could argue that reducing the search space may lead to missing some meaningful rules. We will show in Section 6.4.2 that our generation technique for G creates a representative (reduced) view of the entire search space.

The data structure RuDiK leverages on is a straightforward translation of a KB $\mathcal{K}$ into a directed graph: entities and literals are nodes of the graph, while there is a direct edge from node a to node b for each triple $\langle a, rel, b\rangle \in \mathcal{K}$. Edges are labelled, where the label is the relation rel that connects subject to object in the triple. Figure 6.1 shows a portion of DBPedia [29] that connects two person in a child and parent relationship, along with their dates of birth. The graph represents information of four KB triples (two birth dates, one parent and one child relation).

The body of a Horn Rule can be seen as a path in the graph. W.r.t. Figure 6.1, the body $\texttt{child}(a,b) \wedge \texttt{parent}(b,a)$ corresponds to the path *Clint Eastwood* $\rightarrow$ *Scott Eastwood* $\rightarrow$ *Clint Eastwood*. In Section 6.1.1 we defined valid rules. A valid body of a rule contains target variables a and b at least once, and every other variable at least twice. In a valid body also each atom is transitively connected to every other atom. If we allow navigation of edges in any direction (no matter what the direction of the edge is on the graph), we can easily translate bodies of valid rules to paths on the graph. Given a pair of entities (x, y), a valid body corresponds to a valid path p on the graph that meets the following criteria:

1. p starts at the node x;
2. p touches y at least once;
3. every node can be traversed multiple times;
4. p ends in x, in y, or in a different node that has been touched before.

In other words, given the body of a rule r_{body}, r_{body} covers a pair of entities (x, y) iff there exists a valid path on the graph that corresponds to r_{body}. Therefore, given a pair of entities (x, y), we can generate bodies of all possible valid rules by simply computing all valid paths from x with a standard BFS. Note how the transitively connection between atoms is always guaranteed by the construction property of a path p: for each node n in p there always exists a subpath that connects n to every other node in p. The key point is the ability of navigating each edge in any direction, which basically means turning the original directed graph into an undirected one.

Despite the navigation over an undirected graph, we still need to keep track of the original direction of the edges. This is essential when translating paths to Horn Rule bodies. In fact, if we have a direct edge `rel` from a to b, navigating the edge from a to b produces the atom `rel(`a, b`)`, while navigating the edge from b to a produces the atom `rel(`b, a`)`. These two atoms are different, where the position of variables is determined by the original direction of the edge.

One should notice that for two entities x and y, there might exists infinite valid paths starting from x since every node can be traversed multiple times. Thus we introduce the *maxPathLen* parameter that determines the maximum number of edges in the path. When translating paths to Horn Rules, *maxPathLen* determines the maximum number of atoms that we can have in the body of the rule. This parameter is essential to avoid the discovery of rules with infinite body length.

The main advantage of inspecting just the generation set G is the capability of loading only a small portion of the graph that is currently needed. Given a pair of entities (x, y), we retrieve from the KB all those nodes at distance $maxPathLen - 1$ or less from x or y, along with their edges. Retrieving such nodes and edges can be done recursively: we maintain a queue of entities, and for each entity in the queue we fire a SPARQL query against the KB to get all entities (and edges) at distance 1 from the current entity – we call these queries *single hop queries*. We then add the new found entities to the queue iff they are at distance less than $maxPathLen - 1$ from either x or y and they have not been visited before. The queue is initialised with x and y. By doing so we retrieve a small portion of the entire KB, the only one needed to discover rules that cover (x, y). We will show in the experimental section that SPARQL engines are very fast at executing single hop queries.

The generation of the universe of all possible rules for G is then straightforward: for each element $(x, y) \in G$, we construct the portion of the graph as described above and compute all valid paths starting from x. Computing paths for every example in G implies also computing the coverage over G for each rule. The coverage of a rule r is simply the number of elements in G where there exists a path that is equivalent to r_{body}. Our discovery technique will also generate single-instance rules: rules that cover only one example $(x, y) \in G$ by instantiating target variables a and b in the rule with x and y. Once the universe of all possible rules has been generated (along with coverages over G), computing coverage and unbounded coverage over V is just a matter of executing two SPARQL queries against the KB for each rule in the universe. We will show in Section 6.3 how some queries can be avoided, as well as there is no need of enumerating all possible rules in the universe as some of them will never be part of the final solution.

Clearly, the size of G has a direct impact on the search space and hence on the running time. Since we generate all valid rules for each example in G, the search space grows roughly linearly with the size of G. If we could know a-priori the minimum subset of examples that lead to the generation of all valid rules, then we could use only those few examples. A future direction we will work on exploits exactly this point: how to select a small number of representative examples in order to reduce the size of G, without significantly affecting the quality of the output.

6.2.1 Input Examples Generation

A crucial role in our approach is played by the two input sets G and V, used to generate and validate rules. We will show that discovering negative rules is the dual problem of discovering positive rules, therefore we assume to be in the setting of generating positive rules. We will explain how to switch to the negative setting at the end of this section.

Given an input KB $\mathcal{K}$ and a predicate $rel \in \mathcal{K}$, we automatically build a generation set G and a validation set V as follows. G consists of positive examples for the target predicate rel. It is generated as all pairs of entities (x, y) such that $\langle x, rel, y\rangle \in \mathcal{K}$ – if the target predicate is `child`, it consists of all pairs of entities in a child relation. V instead consists of counter (negative) examples for the target predicate and it is slightly more complicated to generate, since the closed world assumption does not longer hold in KBs. Differently from classic database scenarios, we cannot assume that what is not stated in a KB is false (closed world assumption). Because of large incompleteness, everything that is not stated in a KB is *unknown* rather false. In order to generate negative examples, we make use of a popular technique for KBs: *Local-Closed World Assumption* (*LCWA*) [57, 78]. LCWA states that if a KB contains one or more object values for a given subject and predicate, then it contains all possible values (if a KB contains one or more children of Clint Eastwood, then it contains all possible children). This is definitely true for *functional* predicates (predicates such as `capital` where the subject can have at most one object value), while it might not hold for non-functional predicates (e.g., `child`). KBs contain many non-functional predicates, we therefore extend the definition of LCWA by considering the dual aspect: if a KB contains one or more subject values for a given object and predicate, then it contains all possible values.

To generate negative examples we then take the union of the two LCWA aspects: for a given predicate rel, a negative example is a pair (x, y) where either x is the subject of one or more triples $\langle x, rel, y' \rangle$ with $y \neq y'$, or y is the object of one or more triples $\langle x', rel, y \rangle$ with $x \neq x'$. As an example, if $rel = \texttt{child}$, a negative example is a pair (x, y) such that x has some children in the KB that are not y, or y is the child of someone that is not x. By considering the union of the two LCWA aspects we do not restrict predicates to be functional (such as in [78]).

The number of negative examples generated with the above technique could easily explode (for a target `child` predicate it is nearly the cartesian product of all the people having a child with all the people having a parent). In order to apply the same approach for positive and negative rules discovery, we require G and V to be of comparable sizes. We therefore introduce a further constraint, that significantly shrinks the size of negative examples: given a pair of entities (x, y), x must be connected to y via a predicate that is different from the target predicate. In other words, given a Kb $\mathcal{K}$ and a target predicate rel, (x, y) is a negative examples if $\langle x, rel', y \rangle \in \mathcal{K}$, with $rel' \neq rel$. This intuition exploits the LCWA for predicates rather than for entities. If a KB contains a relation between two entities x and y, then it contains all possible relations between x and y. If x and y are in a relation that is not the target predicate, then most likely x and y are not connect by the target predicate in the real world. This further restriction has multiple advantages: on the one hand it makes the size of V of the same order of magnitude of G (see Section 6.4), on the other hand it guarantees the existence of a path between x and y, for every $(x, y) \in V$. For positive examples, the existence of a path was already guaranteed since pairs in G are always connected by at least one predicate, the target one.

Eventually V is generated with the intersection of the two constraints defined above. A negative example (x, y) for the target predicate `child` has the following characteristics: *(i)* x and y are not connected by a `child` predicate; *(ii)* either x has one or more children (different from y) or y has one or more parents (different from x); *(iii)* x and y are connected by a predicate that is different from `child`.

Often KBs use same predicates to connect different semantic categories. We may find that a `child` predicate is used not only to connect two entities of type person, but also two companies to denote a ownership relation. In order to enhance the quality of the input examples and avoid cases of mixed unrelated semantic categories, we introduce the *type* restriction when generating G and V. The type restriction requires that for every example pair (a, b) belonging to either G or V, a is always of the same type and b is always of the same type. All modern KBs include entity types (often through the `rdf:type` statement), therefore we make use of this information. For example, G and V for a target `child` predicate are two set of pairs, where each pair consists in two entities of type person. The type information can be manually provided or, as we will see in Section 6.4.2, we can automatically compute it from the KB.

In the dual problem of negative rules discovery our approach remains unchanged, we just switch the role of G and V. The generation set becomes V (negative examples), while the validation set becomes G (positive examples). Now it should be more clear why we introduced the second constraint of LCWA on predicates. Since V becomes the generation set in the negative rules setting, V must be small in size and it must guarantee the existence of a path between pairs of entities in each example. Eventually, we underline that our approach is independent on how G and V are generated: they could also be manually crafted by some domain experts, which would require additional manual effort.

To the best of our knowledge, RuDiK is the first approach that is capable of discovering both positive and negative rules in KBs. Previous works have put their focus either on the positive setting [15, 78] (TODO: cite Ontological Pathfinding), or on the negative one (TODO: cite Data Lakes Sigmod). However, none of these techniques can be easily adapted to work in the dual scenario.

6.2.2 Literals and Constants

We defined our target language in Section 6.1.1 which, other than normal predicate atoms, includes literals comparison. The scope of literals comparison is to enrich the language with smarter comparisons among literal values other than equalities, such as greater than or less than. In order to discover such kind of atoms, the KB graph must contain edges that connect literal values with one (or more) symbol from $\{<, \leq, \neq, >, \geq\}$. As an example, Figure 6.1 should contain an edge '$<$' from node "*March 31, 1930*" to node "*March 21, 1986*". Unfortunately, the original KB does not contain this kind of information, and creating such comparisons among all literals in the KB is unfeasible.

Once again, the use of a generation set G is the key point to introduce literals comparison. Since we discover paths for a pair of entities from G in isolation, thus the size of a graph for a pair of entities is relatively small, we can afford to compare all literal values within a single example graph. This implies the creation of a quadratic number of edges w.r.t. the number of literals in the graph. We will show in the experimental section that within a single example graph the number of literals is usually relatively small, thus the quadratic comparison affordable. Modern KBs include three types of literals: numbers, dates, and strings. Besides equality comparisons, we add '$>$','$\geq$','$<$','$\leq$' relationships between numbers and dates, and $\neq$ between all literals. These new relationships are treated as normal atoms: $x \geq y$ is equivalent to $\texttt{rel}(x, y)$, where `rel` is equal to $\geq$. Once we added artificial edges for every input example graph, the discovery of literal comparisons is equivalent to discover normal predicate atoms.

Furthermore, we noticed that '$\neq$' relation could be useful for entities as well other than literals. Think about the following negative rule:

$$\texttt{bornIn}(a, x) \wedge x \neq b \Rightarrow \neg\texttt{president}(a, b)$$

The rule states that if a person a is born in a country that is different from b, then a cannot be president of b. The rule holds for most of the countries in the world. To consider inequalities among entities, we could add artificial edges among all pairs of entities in the graph. This strategy however, despite being inefficient for the high number of edges to add, would lead to many meaningless rules. We noticed that it is reasonable to compare two entities only when they are of the same type, e.g., they belong to the same conceptual category. As with the generation of G and V, we make use of `rdf:type` triples. We add an artificial inequality edge in the input example graph only between those pairs of entities of the same type. In the above rule it is reasonable to compare x and b because they are both countries.

As a last extension of our language, we also discover rules with constants (entities). For a given rule r, we promote a variable v in r to an entity e iff for every $(x, y) \in G$ covered by r, v is always instantiated with the same value e. Suppose that for the above negative rule for president, all examples in G are people connected to the country "*U.S.A.*". We can then promote variable b to "*U.S.A.*", generating the rule:

$$\texttt{bornIn}(a, x) \wedge x \neq U.S.A. \Rightarrow \neg\texttt{president}(a, U.S.A.)$$

6.3 A^* Greedy Algorithm

We introduce in RuDiK a greedy approach to solve the approximate version of the discovery problem (Section 6.1.4). The algorithm combines two phases: *(i)* it solves the set cover problem with a greedy strategy; *(ii)* it discover new rules by navigating the graph in a A^* search fashion, allowing the pruning of unpromising paths.

6.3.1 Marginal Weight

In Section 6.1.3, we defined the weight associated with a set of rules R as follows:

$$w(R) = \alpha \cdot (1 - \frac{\mid C_R(G) \mid}{\mid G \mid}) + \beta \cdot (\frac{\mid C_R(V) \mid}{\mid U_R(V) \mid})$$

Our goal is to discover a set of rules that covers as many elements as possible in G, and as few elements as possible in V. We follow the intuitions behind the greedy algorithm for weighted set cover by defining a *marginal weight* for rules that are not yet included in the solution [43].

Definition 10. Given a set of rules R and a rule r such that $r \notin R$, the marginal

Algorithm 5: Greedy Rules Selection.

```
   input : $G$ – generation set
   input : $V$ – validation set
   input : $R$ – universe of rules
   output: $R_{opt}$ – greedy set cover solution
 1 $R_{opt} \leftarrow \emptyset$;
 2 $r \leftarrow \underset{w_m(r)}{\text{argmin}}(r \in R)$;
 3 repeat
 4     $R_{opt} \leftarrow R_{opt} \cup \{r\}$;
 5     $R \leftarrow R \setminus \{r\}$;
 6     $r \leftarrow \underset{w_m(r)}{\text{argmin}}(r \in R)$;
 7 until $R = \emptyset \vee C_{R_{opt}}(G) = G \vee w_m(r) \geq 0$;
 8 if $C_{R_{opt}}(G) \neq G$ then
 9     $R_{opt} \leftarrow R_{opt} \cup \texttt{singleInstanceRule}(G \setminus C_{R_{opt}}(G))$;
10 return $R_{opt}$
```

weight of r w.r.t. R is defined as:

$$w_m(r) = w(R \cup \{r\}) - w(R) = -\alpha \cdot \frac{| C_r(G) \setminus C_R(G) |}{| G |} + \beta \cdot \left(\frac{| C_{R\cup\{r\}}(V) |}{| U_{R\cup\{r\}}(V) |} - \frac{| C_R(V) |}{| U_R(V) |}\right)$$

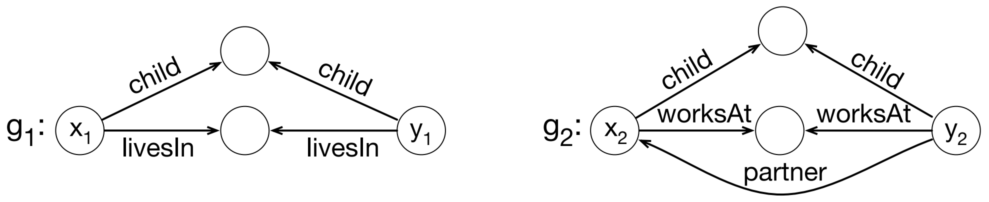
The marginal weight quantifies the total weight increase that we would have by adding r to an existing set of rules R. In other words, the marginal weight indicates the contribution of r to R in terms of new elements covered in G and new elements unbounded covered in V. Due to the first negative part, $w_m(r) \in [-\alpha, +\beta]$. Since the set cover problem aims at minimising the total weight, we would never add a rule to the solution if its marginal weight is greater than or equal to 0. Algorithm 5 shows the straightforward greedy rules selection procedure. The algorithm takes as input the generation set G, the validation set V, and the universe of all possible rules R. The set of output rules R_{opt} is first assigned to an empty set. At each iteration, the algorithm picks from R the rule r with the minimum marginal weight, and it adds r to R_{opt}. r is then removed from R. The algorithm stops when one of the following termination conditions are met: 1. R is empty – all the rules have been included in the solution; 2. R_{opt} covers all elements of G; 3. the minimum marginal weight is greater than or equal to 0 – among the remaining rules in R, none of them has a negative marginal weight, hence the current solution is the one with the minimum weight. If the second termination condition is not met, there may exist examples in G that are not covered by R_{opt}. In such a case the algorithm will augment R_{opt} with single-instance rules (rules that cover only one example), one for each element in G not covered by R_{opt}.

Since the coverage of a rule is always contained in its unbounded coverage, the marginal weight is greater than or equal to 0 whenever the rule does not cover new elements in G and does not unbound cover new elements in V. More specifically, a rule r has a negative marginal weight iff the sum of its new elements covered in G with its new elements unbounded covered in V is strictly greater than its new elements covered in V multiplied by some γ, where γ depends on how we set α and β.

The greedy solution guarantees a `log(`k`)` approximation to the optimal solution [43], where k is the largest number of elements covered in G by a rule in R – k is at most $|G|$. If the output rules in the final solution cover disjoint set of G, then the greedy solution coincides with the optimal one.

6.3.2 A^* Graph Traversal

Algorithm 5 assumes that the universe of rules R has been generated, so that we can iteratively pick the rule with minimum marginal weight. In order to generate the universe of all possible rules, we need to traverse all valid paths from a node x to a node y, for every pair (x, y) in the generation set G. In this section we will analyse the following aspect: is it essential to generate all possible paths for every example in G?



Example 20. Consider the scenario where we are mining positive rules for the target predicate `spouse`. The generation set G includes two examples g_1 and g_2, where the above figure shows the KB graph for the two examples. Assume for simplicity that all rules in the universe have the same coverage and unbounded coverage over the validation set V. One of the plausible rule is r = `child(`x, v_0`)` $\wedge$ `child(`y, v_0`)` $\Rightarrow$

`spouse`(x, y), stating that entities x and y with a common child are married. Looking at the KB graph, r covers both g_1 and g_2 – in both g_1 and g_2 there exists a path that corresponds to r_{body}. Since all rules have the same coverage and unbounded coverage over V, if we include r in the solution before inspecting other potential rules, then there is no need to generate any other rule. In fact any other plausible rule will not cover new elements in G, therefore their marginal weights will be greater than or equal to 0. Hence any other rule will not be part of the solution. The navigation (and creation) of edges `livesIn` in g_1, `worksAt` in g_2, and `partner` in g_2 becomes worthless.

Based on the above observation, we avoid the generation of the entire universe R, but rather we consider at each iteration the most promising path on the graph. The intuition is the same behind the A^* graph traversal algorithm [89]. Given an input weighted graph, A^* computes the smallest cost path from a starting node s to an ending node t. At each iteration, A^* maintains a queue of partial paths starting from s, and it expands one of these paths based on an *estimation* of the cost still to go to t. The path with the best estimation is expanded and added to the paths queue. The algorithm keeps iterating until one of the partial paths reaches t. RuDiK discovers rules with a similar technique. For each example $(x, y) \in G$, we start the navigation from x. We keep a queue of not valid rules (Section 6.1.1), and at each iteration we consider the rule with the minimum marginal weight, which corresponds to equivalent paths in the example graphs. We expand the rule by following edges on the graphs, and we add the new founded rules to the queue of not valid rules. Differently from A^*, we do not stop when a rule (path) reaches the node y. Whenever a rule becomes valid, we add the rule to the solution and we do not expand it any further. The algorithm keeps looking for plausible paths until one of the termination conditions of Algorithm 5 is met.

A crucial point in A^* is the definition of the estimation cost. In particular, if we want to guarantee the solution to be optimal, the estimation must be *admissible*, i.e., the estimation cost must be less than or equal to the actual cost. As an example, when searching for the shortest route on a map, an admissible estimation might be the straight-line distance to the goal for every node, since that is physically the smallest possible distance between any two points. In our setting, given a rule that is not yet valid and needs to be expanded, we need to define an admissible estimation of the marginal weight.

Definition 11. Given a rules $r = A_1 \wedge A_2 \cdots A_n \Rightarrow B$, we say that a rule r' is an *expansion* of r iff r' has the form $A_1 \wedge A_2 \cdots A_n \wedge A_{n+1} \Rightarrow B$. In other words, r' is generated by adding a new atom to the body of r.

Given a rule r, expanding r means adding a new atom to the body of r. From the graph traversal point of view, expanding r means traversing one further edge on the path defined by r_{body}. In order to guarantee the optimality condition, the estimated marginal weight for a rule r that is not valid must be less than or equal to the actual weight of any valid rule that is generated by expanding r. Given a rule and some expansions of it we can derive the following:

Lemma 6. *Given a rule r and a set of pair of entities E, then for each r' expansion of r, $C_{r'}(E) \subseteq C_r(E)$ and $U_{r'}(E) \subseteq U_r(E)$.*

The above Lemma states that the coverage and unbound coverage of a rule r' expansion of r are respectively contained in the coverage and unbound coverage of r. In fact expanding a rule means adding a new atom to the body of the rule, hence making the rule more selective. This is equivalent of adding an 'AND' condition to a SQL query q: the new result will obviously be a subset of the result obtained with q.

We recall that the merginal weight of a rule r w.r.t. a solution R is defined as follow:

$$w_m(r) = -\alpha \cdot \frac{\mid C_r(G) \setminus C_R(G) \mid}{\mid G \mid} + \beta \cdot (\frac{\mid C_{R \cup \{r\}}(V) \mid}{\mid U_{R \cup \{r\}}(V) \mid} - \frac{\mid C_R(V) \mid}{\mid U_R(V) \mid})$$

The only positive contribution to marginal weights is given by $|C_{R \cup \{r\}}(V)|$. $|C_{R \cup \{r\}}(V)|$ is equivalent to $|C_R(V)| + |C_r(V) \setminus C_R(V)|$, thus if we set $|C_r(V) \setminus C_R(V)| = 0$ for any r that is not valid, we guarantee an admissible estimation of the marginal weight. More simply, we estimate the coverage over the validation set to be 0 for any rule that can be further expanded, since expanding the rule may bring the coverage to 0.

Definition 12. Given a rule r and a set of rules R, we define the *estimated marginal weight* of r as:

$$w^*_m(r) = \begin{cases} -\alpha \cdot \frac{|C_r(G) \setminus C_R(G)|}{|G|} + \beta \cdot (\frac{|C_R(V)|}{|U_{R \cup \{r\}}(V)|} - \frac{|C_R(V)|}{|U_R(V)|}) & \text{if } r \text{ is not valid} \\ w_m(r) & \text{if } r \text{ is valid} \end{cases}$$

The estimated marginal weight for a valid rule is equal to the actual marginal weight since a valid rule will not be considered for expansion – it corresponds to reaching the ending node in the A^* algorithm. Given Lemma 6, we can easily see that $w^*_m(r) \leq w^*_m(r')$, for any r' expansion of r. Thus our marginal weight estimation is admissible.

RuDiK uses the concept of *frontier nodes* for a rule r ($N_f(r)$). Given a rule r, $N_f(r)$ contains the last visited nodes in the paths that correspond to r_{body} from every example graphs covered by r. As an example, given $r_{body} = \texttt{child}(x, v_0)$, $N_f(r)$ contains all the entities v_0 that are children of x, for every $(x, y) \in G$ – v_0 is the last visited node in the path. Expanding a rule r implies navigating a single edge from any frontier node. Algorithm 6 shows the modified set cover version that includes A^*-like rules generation. The set of frontier nodes is initialised with starting nodes x, for every $(x, y) \in G$ (Line 2). The algorithm maintains a queue of rules Q_r, from which it chooses at each iteration the rule with minimum approximated weight. The function `expandFrontiers` retrieves from the KB all nodes (along with edges) at distance 1 from frontier nodes and returns the set of all rules generated by this one hop expansion. Such expansions are computed with single-hop SPARQL queries. Q_r is therefore initialised with all rules of length 1 starting at x (Line 3). In the main loop, the algorithm checks if the current best rule r is valid or not. If r is valid, r is added to the output and it is not expanded (Line 8). If r is not valid, r is expanded iff the length of its body is less than *maxPathLen* (Line 10). There is no point in expanding rules with body greater than or equal to *maxPathLen* since such rules are not allowed in the output. The termination conditions and the last part of the algorithm are the same of Algorithm 5.

The simultaneous rules generation and selection of Algorithm 6 brings multiple benefits. First of all, we do not generate the entire graph for every example in G.

Algorithm 6: RuDiK Rules Discovery.

```
input  : $G$ – generation set
input  : $V$ – validation set
input  : $maxPathLen$ – maximum rule body length
output: $R_{opt}$ – greedy set cover solution
1  $R_{opt} \leftarrow \emptyset$;
2  $N_f \leftarrow \{x|(x,y) \in G\}$;
3  $Q_r \leftarrow$ expandFrontiers($N_f$);   // SPARQL queries
4  $r \leftarrow \underset{w^*_m(r)}{\text{argmin}}(r \in Q_r)$;
5  repeat
6      $Q_r \leftarrow Q_r \setminus \{r\}$;
7      if isValid($r$) then
8          $R_{opt} \leftarrow R_{opt} \cup \{r\}$;
9      else
           // rules expansion
10         if length($r_{body}$) $< maxPathLen$ then
11             $N_f \leftarrow$ frontiers($r$);
12             $Q_r \leftarrow Q_r \cup$ expandFrontiers($N_f$);   // SPARQL queries
13     $r \leftarrow \underset{w^*_m(r)}{\text{argmin}}(r \in Q_r)$;
14 until $Q_r = \emptyset \vee C_{R_{opt}}(G) = G \vee w^*_m(r) \geq 0$;
15 if $C_{R_{opt}}(G) \neq G$ then
16     $R_{opt} \leftarrow R_{opt} \cup$ singleInstanceRule($G \setminus C_{R_{opt}}(G)$);
17 return $R_{opt}$
```

Nodes and edges are generated *on demand*, whenever the algorithm requires their navigation (Line 12). If the initial part of a path is not promising according to its estimated weight, the rest of the path will never be materialised since there is no need to navigate it. Materialising the graph is an expensive task, since we need to query the disk in order to retrieve target nodes and edges. Rather than materialising the entire graph and then traversing it, we propose a solution that gradually materialises parts of the graph whenever they are needed for navigation (Lines 3 and 12). Secondly, the weight estimation could lead to pruning unpromising rules. If a rule does not cover new elements in G and does not unbound cover new elements in V, then its estimated marginal weight is 0 (Definition 12). A rule with 0 marginal weight is never picked as best rule to be expanded, and if it is the algorithm terminates (due to one of the termination conditions). This implies that a rule with 0 estimated marginal weight is pruned away from the search space, as we will never generate rules from its expansion.

The partial materialisation of the graph and the pruning of the search space have a significant impact on the algorithm's resources. On the one hand, we noticed the running time is halved in the worst case and sometimes it is up to ten times faster. On the other hand, we can also significantly decrease the amount of memory needed, since we load sub-portions of the graph. Consider the extreme case where there exists a rule r that covers all elements in G, unbound covers all elements in V and does not cover any element of V. In such a case, Algorithm 6 will output the optimal solution by just materialising paths on the example graphs that correspond to r_{body} with $\mathcal{O}\left(l \cdot |G|\right)$ SPARQL queries, where l is the length of r_{body}.

6.4 Experiments

TODO: add EXPERIMENTS TO AN ONLINE APPENDIX AND MENTION IT HERE

We carried out an extensive experimental evaluation of our rules discovery approach. We grouped the evaluation into 4 main sub-categories: *(i)* a first set of experiments aims at demonstrating the quality of our output, both for negative and positive rules; *(ii)* a second set of experiments compares our method with state-of-the-art systems; *(iii)* in the third set of experiments we outline the applicability of rules discovery by enhancing machine learning algorithms; *(iv)* in the last set of experiments we discuss internal system settings and some KB properties.

Settings. We evaluated our approach over several popular KBs. For each KB, we downloaded the most up-to-date core facts and loaded them into our SPARQL query engine. We experimented several SPARQL engines, including Jena ARQ[5], OWLIM Lite[6], and RDF-3x[7]. We also implemented a naïve relational database solution with PostgreSQL. Eventually we opted for OpenLink Virtuoso[8], as it was the fastest among all the solutions. Virtuoso took on average 20 minutes to load a medium size KB (i.e., 10 GB) into its store, and around 100 milliseconds to execute a single hop query. All experiments are run on a iMac desktop with an Intel quad-core i5 at 2.80GHz with

[5]https://jena.apache.org/documentation/query/
[6]https://confluence.ontotext.com/display/OWLIMv54/OWLIM-Lite+Installation
[7]https://code.google.com/archive/p/rdf3x/
[8]http://virtuoso.openlinksw.com/

16 GB RAM. We run Virtuoso server with its SPARQL query endpoint on the same machine, optimised for a 8 GB available RAM.

Our method needs the two input parameters α and β of Equation 6.1. α measures the importance of the coverage over the generation set, while β measures the coverage over the validation set. In other words, a high α privileges recall over precision, while a high β gives more importance to precision. We can afford a high α and a low β when the input KB is accurate and complete. We will show that KBs contain many errors and missing information, therefore we set $\alpha = 0.3$ and $\beta = 0.7$. Increasing α and decreasing β means a higher number of output rules with a lower accuracy.

We also set the $MaxPathLen$ parameter to 3. This number represents the maximum number of atoms that we can have in the body of a rule. We will show in Section 6.4.4 that increasing this parameter does not bring any benefits, as body rules longer than 3 atoms start to be very complicated and not insightful.

Evaluation Metrics. RuDiK discovers rules for a given target predicate. For each KB, we chose 5 representative predicates as follows: we first ordered predicates according to descending popularity (i.e., number of triples having that predicate), and then we picked the top 3 predicates for which we knew there existed at least one meaningful rule, and other 2 top predicates for which we did not know whether some meaningful rules existed. We repeated the procedure for each input KB, and for positive and negative rules. Despite working one predicate at time, RuDiK can also discover rules for the entire KB by listing all predicates in the KB, and discover rules for each of them. We will show in Section 6.4.2 how this can be achieved.

Positive rules are very useful to enrich the KB by discovering new facts. Since we are generating new data, we cannot evaluate the output over the existing data. Inspired by [78], we proceed as follows: we run the algorithm over the KB, and for each output rule we generate all new predictions that are not already in the KB (we execute the body of the rule against the KB and remove all those pairs that are already connected by the target predicate in the KB). As an example, for the rule $\texttt{spouse}(b, a) \Rightarrow \texttt{spouse}(a, b)$, we retrieve all pairs (b, a) such that b is spouse with a but a is not spouse with b in the KB. If the rule is universally correct (like the previous example), we mark all the new predictions as true. If the rule is unknown, we randomly sampled 30 new predictions and manually check them against the Web. The *precision* of the rule is then computed as the ratio of true predictions out of true and false predictions.

Negative rules are slightly more complicated to evaluate. In fact, despite KBs are usually incomplete, a large percentage of the data not stated in a KB is false – a very small fraction of the cartesian product of all the people in a KB will be actually married. Therefore negative rules will always discover many correct negative facts. However, negative rules are a great means to discover errors in the KB, and we leverage this aspect to evaluate them. For each discovered rule, we retrieve from the KB pairs of entities for which the body of the rule can be instantiated over the KB and that are also connected by the target predicate. As an example, for the rule `child`$(a, b) \Rightarrow \neg$`spouse`(a, b), we retrieve all those pairs (a, b) such that b is `child` of a and a is `spouse` with b. We call these generated pairs of entities *potential errors*. Similarly to positive rules, whenever a rule is universally correct we mark all its potential errors as true, whereas if the rule is unknown we manually check 30 sampled potential errors. The final precision of a rule is computed as actual errors divided by potential errors. Furthermore, a potential error is evaluated as actual error whenever a single atom in the rule is incorrect, no matter if the atom is in the body or the head of the rule. For the negative rule `child`$(a, b) \Rightarrow \neg$`spouse`(a, b), a specific instantiation (a, b) is an actual error if either b is not an actual `child` of a, or b is not an actual `spouse` of a. As we will explain further below, negative rules cannot point exactly where the error is, but they give a hint that something is wrong and needs to be further inspected.

6.4.1 Rules Discovery Accuracy

The first set of experiments aims at evaluating the accuracy of discovered rules over the 3 most popular and widely used KBs: DBPedia [29], Yago [132], and Wikidata [137]. For each KB we downloaded the most recent version and selected core facts, facts about people, geolocations and transitive `rdf:type` facts. Wikidata provides only the entire dump, therefore we just eliminated from it no-english literal values.

Table 6.1: Dataset characteristics.

KB	*Version*	*Size*	*#Triples*	*#Predicates*
DBPedia	3.7	10.056GB	68,364,605	1,424
Yago	3.0.2	7.82GB	88,360,244	74
Wikidata	20160229	12.32GB	272,129,814	4,108

Table 6.1 shows the characteristics of the 3 KBs.

As the figure shows, the size of the KB is relevant. Loading the entire KB into main memory is not feasible unless we have high memory availability (TODO: cite the two SIGMOD16 papers), or we reduce the KB by eliminating facts such as `rdf:type` or literals [78]. We propose an approach that is disk-based where only a small portion of the KB is loaded into main memory, such that we can discover rules on any size KB with a normal machine.

6.4.1.1 Positive Rules Discovery

We first evaluate the precision of positive rules for the top 5 predicates on the 3 KBs. The number of new induced facts varies significantly from rule to rule – a rule with literals comparison will produce a very high number of facts. In order to avoid the precision to be dominated by such rules, we first compute the precision for each rule as explained above, and then we average values over all induced rules. Table 6.2 reports precision values, along with predicates average running time.

Our first observation is that the more accurate is the KB, the better is the quality of induced rules. Wikidata contains very few errors, since it is manually curated and every triple is manually checked by different individuals before being inserted. DBPedia and Yago instead are automatically generated by extracting information from the Web, hence their quality is significantly lower. Discovering perfect positive rules is a hard task, mostly because there is no guarantee of the existence of valid

Table 6.2: Positive Rules Accuracy.

KB	*Avg. Running Time*	*Precision*
DBPedia	34min, 56sec	**63.99**%
Yago	59min, 25sec	**62.86**%
Wikidata	2h, 21min, 34sec	**73.33**%

negative examples. A striking example in this direction is one of the rule induced for `founder` in DBPedia. Our approach discovers that if a person is born in the same place where a company is founded, then the person is the founder of the company. The rule is obviously wrong, as there are many people who are born in the same place of a company and have not founded the company. However this rule has a very high coverage over the generation set (many companies' founders founded their company in their birth place), and a very low coverage over the validation set – indeed among the cartesian product of all the people and companies, a very small fraction includes people and companies born and founded in the same place. Despite such hard cases, our approach is always capable of producing correct rules for those predicates for which we knew there existed some valid rules. Cases like `academicAdvisor`, `child`, and `spouse` have a precision above 95% in all of the KBs, and final precision values are brought down by few predicates where meaningful rules probably do not exist at all.

The running time is influenced by different factors. First of all the size of the KB has obviously a huge impact, as RuDiK is slower in Wikidata which is the biggest KB. Not only the number of triples is relevant, but also the different number of predicates. In fact the more predicates we have in the KB, the more alternative paths we observe when traversing the graph, hence a bigger search space. The second relevant aspect is the target predicate involved. We noticed that some kind of entities have a huge number of outgoing and incoming edges ("*United States*" in Wikidata is connected to more than 600K entities). When the generation set includes such type of entities, the navigation of the graph is slower as we need to traverse a high number of edges. This is what happens in Yago, where the most popular predicates are `isLeaderOf` and `exports`. Eventually the *maxPathLen* parameter also has a big say in the final running time. The longer the rule, the bigger is the search space. We will show in the next Section how we can be much faster if we set to 2 atoms the maximum length of the rule. In the next chapter we will discuss some future directions to significantly cut down the running time based on the above observations.

Table 6.3: Negative Rules Accuracy.

KB	*Avg. Run Time*	*# Potential Errors*	*Precision*
DBPedia	19min, 40sec	499	**92.38%**
Yago	10min, 40sec	2,237	**90.61%**
Wikidata	1h, 5min, 38sec	1,776	**73.99%**

6.4.1.2 Negative Rules Discovery

Negative rules are very useful to discover inconsistencies in the KB. We evaluated negative rules as the percentage of correct errors discovered for the top 5 predicates in each KB. Table 6.3 shows, for each KB, the total number of potential erroneous triples discovered with negative rules, whereas the precision is computed as the percentage of actual errors among potential errors.

Negative rules generally have better accuracy than positive ones. This is mostly due to the completeness of the validation set: for negative rules the validation set is the universe of all possible counter examples stated in the KB, whereas for positive rules the validation set is just a small fraction of it. Therefore negative rules are usually better validated than positive ones. Wikidata shows lower numbers just because it does not contain as many errors as DBPedia and Yago: even though discovered rules are almost correct, the percentage of actual errors identified is lower in Wikidata. For example, we identify the same rule that two people with same gender cannot be married both in Yago and Wikidata. Such rule retrieves errors in Yago with 94% accuracy, while the accuracy in Wikidata for the same rule is 57%. Yago is the KB with the highest number of errors. As an example, there are $9,057$ cases in the online Yago where a child is born before her parent. We cannot point exactly where the error is – there could be an error in one of the birth dates or an error in the parent relation – but we can affirm that at least one of these values is wrong and send them to human evaluators to spot the inconsistency.

Differently from positive rules, literals play a vital role in discovering negative rules. In fact in many cases correct negative rules rely on temporal aspects in which something cannot happen before/after something else. Temporal information are usually expressed through dates, years, or other primitive types that are represented as literal values in KBs.

As a last observation, discovering negative rules is usually faster than discovering positive rules. This is mostly due to the time we spend executing validation coverage queries. Whenever we discover a rule that respect our language bias (Section 6.1.1), we execute the body of the rule against the KB with a SPARQL query to compute its coverage over the validation set. These queries are faster for negative rules since the validation set is just all the entities connected by the target predicate, whereas in the positive case the validation set corresponds to counter examples described in Section 6.2.1, which are usually more complex to evaluate for standard SPARQL engines.

Table 6.3 shows some interesting cases, for both correct and incorrect rules. The example rules show the full power of our language, including literals, smart literals comparisons, entities inequalities and constants. As previously mentioned, the extension including literals comparisons and entities inequalities gives a significant boost in accuracy for negative rules, while it is rarely used in discovery positive rules.

Rule	*KB*	*Precision*
$\texttt{notableWork}(b,a) \Rightarrow \texttt{creator}(a,b)$	WIKIDATA	**100%**
$\texttt{nationality}(a,v_0) \wedge \texttt{nationality}(b,v_0) \wedge \texttt{notableStudent}(b,a) \Rightarrow \texttt{academicAdvisor}(a,b)$	DBPEDIA	**100%**
$\texttt{hasChild}(v_0,b) \wedge \texttt{isMarriedTo}(a,v_0) \Rightarrow \texttt{hasChild}(a,b)$	YAGO	50%
$\texttt{activeYearsEndDate}(a,v_0) \wedge \texttt{activeYearsStartDate}(b,v_0) \Rightarrow \texttt{successor}(a,b)$	DBPEDIA	0%
$\texttt{inception}(a,v_0) \wedge \texttt{dateOfBirth}(b,v_1) \wedge v_1 > v_0 \Rightarrow \neg\texttt{founder}(a,b)$	WIKIDATA	**100%**
$\texttt{country}(a,England) \wedge \texttt{country}(b,United\_Kingdom) \wedge England \neq United\_Kingdom \Rightarrow \neg\texttt{county}(a,b)$	DBPEDIA	**100%**
$\texttt{actedIn}(a,b) \Rightarrow \neg\texttt{wroteMusicFor}(a,b)$	YAGO	40%
$\texttt{birthYear}(a,v_0) \wedge \texttt{birthDate}(b,v_1) \wedge v_0 < v_1 \Rightarrow \neg\texttt{academicAdvisor}(a,b)$	DBPEDIA	28.6%

Table 6.4: Interesting Output Rules.

6.4.2 Comparative Evaluation

We compared the performance of our rules discovery method against AMIE [78], the state-of-the-art system in discovery Horn Rules from KBs.

AMIE is a rules discovery system designed to discover positive rules. It first loads the entire KB into memory, and then discovers positive rules for every predicate in the KB. AMIE lists all the predicates in the KB and inserts each of them as head of the rule. Once the head is filled, the system tries to expand the rule by pivoting on one of the variables of the current predicate and looks for predicates sharing the same variable with high coverage in the KB. The coverage of a rule is penalised with the partial closed world assumption, where the set of negative examples for a given pair (x, y) and a target predicate p is all those pairs where x is connected through p to an entity different from y. Differently from us, AMIE outputs all possible rules that exceeds a given threshold and ranks them according to their coverage function.

Given the in-memory implementation, AMIE cannot handle large KBs. We tried to run it on the KBs of Table 6.1, but the system goes quickly out of memory. Thus we downloaded and used the modified versions of YAGO and DBPEDIA used in their experiments [9]. These versions consist in the core facts of the KB, without literals and `rdf:type` facts. Table 6.5 summarises the characteristic of this dataset.

Table 6.5: AMIE Dataset characteristics.

KB	*Size*	*#Triples*	*#Predicates*	*#*`rdf:type`
DBPEDIA	551M	7M	10,342	22.2M
YAGO	48M	948.3K	38	77.9M

Removing literals and `rdf:type` triples drastically reduce the size of the KB (Table 6.1). Since our approach needs type information (both for the generation of G and V and for the discovery of entities inequality atoms), we run AMIE on its original dataset, while we run our algorithm on the same dataset plus `rdf:type` triples. The last column of Table 6.5 shows how many triples we added to the original AMIE dataset.

Positive Rules. We first compared RuDiK against AMIE on its natural setting: positive rules discovery. AMIE takes as input an entire KB, and discovers rules for every predicate in the KB. We adapted our system to simulate AMIE as follows: we first list all the predicates in the KB, and for each predicate that connects a *subject* to an *object* we computed the most common type for both subject and object. The most common type is computed as the most popular `rdf:type` that is not super class of any other most popular type.

[9]`www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/amie/`

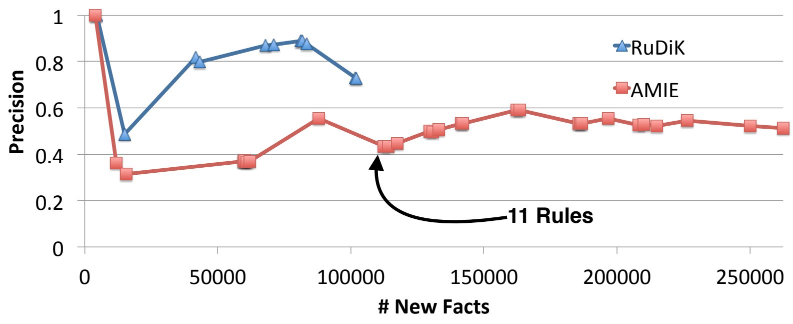


Figure 6.2: Predictions Accuracy on YAGO

After computing type domain and co-domain for each predicate, we run our approach sequentially on every predicate. Furthermore we set the *maxPathLen* parameter to 2, since this is the default setting for AMIE.

AMIE outputs a huge amount of rules along with their scores – 75 output rules in YAGO, and 6090 in DBPEDIA. We followed their experiments setting and picked the first 30 best rules according to their score. We then picked the rules produced by our approach on the head predicate of the 30 best rules output of AMIE. RuDiK is more conservative and produces much less rules than AMIE. We noticed that for every predicate AMIE always discovers more than one rule, while there are several cases where the output of our algorithm is empty since none of the plausible rules has enough support. This results, for instance, in just 11 rules in output on the entire YAGO.

The precision of each rule is computed as described above with a minor modification: whenever a rule is unknown with its new predictions, we first check the existence of the new induced facts in a newer version of the KB. This is possible because the AMIE dataset does not contain the most up to date versions. If a new fact does not appear neither in a newer version nor on the Web, it is evaluated as false.

Figure 6.2 plots the total number of new unique predictions (x-axis) versus the aggregated precision (y-axis) on YAGO. The n-th point from the left represents the total number of predictions and the total precision of these predictions, computed over the first n rules (sorted according to AMIE's score). AMIE produces many more predictions (262K vs 102K), but with a significant lower accuracy. This is due to the high number of rules in output of AMIE, but also to the way these rules are ranked. In

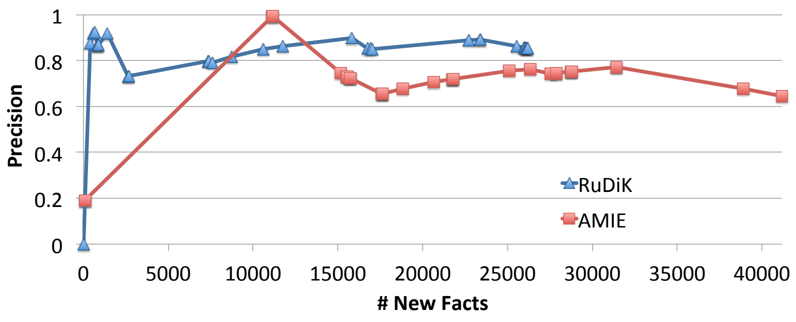


Figure 6.3: Predictions Accuracy on DBPEDIA

fact if we limit the output of AMIE to the best 11 rules (same output of our approach), the final accuracy is still 29% below our approach, with just 10K more predictions. AMIE outputs many good rules that are preceded by meaningless ones in the ranking, and it is not clear how to set a proper k in order to get the best top k rules. RuDiK instead understands that in some cases meaningful rules do not exist, and it outputs something only when it has a strong confidence. This results in a lower number of predictions with a very high accuracy – precision is above 85% before the last rule, with more than 80K predictions. Moreover, our approach can also simulate AMIE if we are more interested in recall. By modifying the α and β parameters, we can obtain a higher number of predictions in output, at the expense of accuracy.

Figure 6.3 shows the same evaluation on DBPEDIA. DBPEDIA has a richer set of relations, therefore also RuDiK is capable of producing 30 rules in output. Despite the same number of rules, once again our approach leads to a lower number of predictions (26K vs 41K) with a significant higher accuracy (85% vs 74%). The only point where AMIE outperforms our approach is when we consider the top 3 rules: the third rule discovered by AMIE is indeed a universally true rule that produces more than 11K correct predictions.

Negative Rules. As a second set of comparative experiments we used AMIE to discover negative rules. AMIE is not designed to work in this setting, and can discover rules only for predicates explicitly stated in the KB. Therefore we proceeded as follows: we sampled the top 5 most popular predicates in each KB and we created for each predicate a set of negative examples as explained in Section 6.2.1. For each negative example we added a new fact to the KB connecting the two entities with the *negation*

Table 6.6: Negative Rules vs AMIE.

	AMIE		**RuDiK**	
KB	*# Errors*	*Precision*	*# Errors*	*Precision*
DBPedia	457	38.85%	148	**57.76**%
Yago	633	48.81%	550	**68.73**%

of the predicate. For example, we added a `notSpouse` predicate connecting each pair of people who are not married according to our negative examples generation technique. We then run AMIE on these new created predicates. The evaluation of negative rules is then carried out as explained before: we generate potential errors in the KB, and we manually evaluated the precision of such errors. Table 6.6 shows the results on the two KBs. RuDiK outperforms AMIE in both cases of almost 20%. This is because for the negation of a predicate, we use the actual predicate as counter examples. AMIE instead is not aware that the actual predicate provides counter examples for the negation of it, hence it is much less precise. In fact the output of AMIE consists in a high number of rules for each predicate (often more than 30), and in many cases AMIE produces same rules for both positive and negative scenarios. As an example, AMIE outputs that if a country a exports a good b, then a imports b and a does not import b.

Despite clearly outperforming AMIE, numbers look significantly lower that the ones showed in Section 6.4.1. This is because we are using the AMIE modified KBs which do not contain literals. As explained earlier, literals play a vital role when discovering negative rules, both in terms of total errors discovered and in terms of precision. Excluding literals is a big disadvantage, and we will emphasise this aspect in detail in Section 6.4.4. (TO DO: check if the internal evaluation actually exists)

Running Time. We report here the running time of AMIE compared to our approach. Note that numbers are different from [78], where AMIE was run on a 48 GB RAM server. AMIE could finish the computation only on Yago 2, while for the other KBs it got stuck without outputting any more rules. When this happened, we stopped the computation after we did not see any new rule in output for more than 2 hours.

Table 6.7 reports the running time on different KBs. The first five KBs are the AMIE modified versions, while Yago 3* is complete Yago, including literals and `rdf:type`. The third column shows the number of predicates for which AMIE was able to produce

Table 6.7: Run Time vs AMIE.

KB	*#Triple*	*#Predicates*	*AMIE*	*RuDiK*	*Types Time*
Yago 2	948.3K	20	30s	18m,15s	12s
Yago 2s	4.1M	26 (38)	> 8h	47m,10s	11s
DBPedia 2.0	7M	904 (10342)	> 10h	7h,12m	77s
DBPedia 3.8	11M	237 (649)	> 15h	8h,10m	37s
Wikidata	8.4M	118 (430)	> 25h	8h,2m	11s
Yago 3*	88.3M	72	-	2h,35m	128s

at least one rule. In some cases AMIE got stuck without producing any rules for some predicates, hence we report the total number of predicates in brackets. For a fair comparison we run our algorithm only on those predicates for which AMIE could produce at least one rule. The fourth and fifth columns report the total running time of the two approaches. Despite being disk-based, RuDiK can successfully complete the task faster than AMIE in all cases, except for Yago 2. This is because of the very small size of the KB, which can easily fits in main memory. However, when we deal with real KBs (Yago 3*), AMIE is not even capable of loading the KB due to out of memory errors. Eventually the last column reports the total time needed to compute `rdf:type` information for each predicate in the KB. Such time is negligible w.r.t. the total running time. The running time justifies our disk-based strategy: RuDiK can successfully discover rules for any size KB on any machine.

Other Systems. We found other available systems to discover rules in KBs. [15] discovers new facts at instance level, hence less generic than our approach. On AMIE Yago 2 KB they can discover 2K new facts with a precision lower than 70%. The best rule we discover on Yago 2 already produces more than 4K facts with a 100% precision. (TODO: CITE Ontological Path Finding) implements AMIE algorithm with a focus on scalability. They do not introduce any novelty in the algorithmic part, but just a clever way of splitting the KB into multiple cluster nodes so that the computation can be run in parallel. The output is the same as AMIE. Eventually we did not compare with classic Inductive Logic Programming systems [55, 113], as these are already significantly outperformed by AMIE both in accuracy and running time.

6.4.3 Machine Learning Application

The main goal of this set of experiments is to prove the applicability of our approach in helping Machine Learning algorithms to provide meaningful training examples. We chose DeepDive [128], a Machine Learning approach to incrementally construct KBs. DeepDive extracts entities and relations from text articles via distant supervision. The key idea behind distant supervision is to use an external source of information (e.g., a KB) to provide training examples for a supervised algorithm. For example, the main showcase in DeepDive extracts mentions of married couples from text documents. In such a scenario DeepDive uses as a first step DBPedia in order to label some pairs of entities as *true* positive (those pairs of married couples that can be found in DBPedia). These labelled examples are then used to construct a factor graph, similar to Markov Logic, that will predict labels on the rest of candidates. Unfortunately, a KB can only provide positive examples. Hence in DeepDive the burden of creating negative examples is left to the user through manual rules definition.

In this set of experiments we will use our negative rules on DBPedia to generate negative examples, and we will compare the output of DeepDive trained with different set of negative examples. We used DeepDive showcase example[10], where the goal is to extract mentions of married people from text articles. DeepDive already provides some negative rules to generate negative examples (e.g., if two people appear in a sentence connected by the words *brother* or *sister* then they are not married). We therefore compare the output of DeepDive using our generated negative examples and the ones generated with DeepDive rules.

Figure 6.4 shows DeepDive accuracy plot run on 1K input documents. The accuracy plot shows the ratio of correct positive predictions over positive and negative predications (y-axis), for each probability output value (x-axis). The dotted blue line represents the ideal situation, where the system finds high number of evidence positive predictions for higher probability output values – when the output probability is 0 there should not be positive predictions. The plot is computed over a test set, while the system is trained over a separated training set. The figure shows 4 lines other than the

[10]`http://deepdive.stanford.edu/`

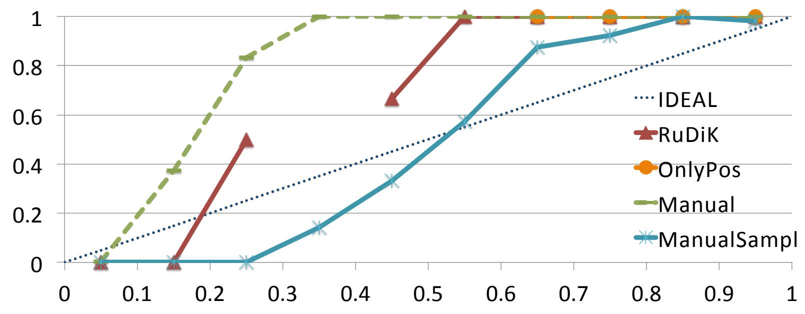


Figure 6.4: DeepDive Application 1K articles

ideal ones. RuDiK is the output of DeepDive using our approach to generate negative examples. `OnlyPos` uses only positive examples from DBPedia, `Manual` uses positive examples from DBPedia and manually defined rules to generate negative examples, while `ManualSampl` uses only a sample of the manually generated negative examples in size equal to positive examples. The first observation is that `OnlyPos` and `Manual` do not provide valid training, as the former has only positive examples and labels everything as true, while the latter has many more negative examples than positive and labels everything as false. `ManualSampl` is the clear winner, while our approach suffers mostly the absence of data: over the input 1K articles, we could find only 20 positive and 15 negative examples from DBPedia.

If we extend the input to 1M articles, things change drastically (Figure 6.5). All the three approaches except `OnlyPos` can successfully drive DeepDive in the training, with the examples provided with RuDiK leading to a slightly better result. This is because of the quality of the negative examples: our negative rules generate representative examples that can help DeepDive in understanding discriminatory features between positive and negative labels. The output of `ManualSampl` and RuDiK are very similar, meaning that we can use our approach to simulate user behaviour and provide negative examples. With manually defined rules the number of generated examples is significantly higher (23K vs 5K generated with RuDiK), however the results are very similar since a small number of significant examples is enough to provide complete evidence for the training. This confirms the main finding of this experiment: as long as we have an external source of information with a decent coverage over the input articles, users do not need to worry about providing rules to generate negative examples.

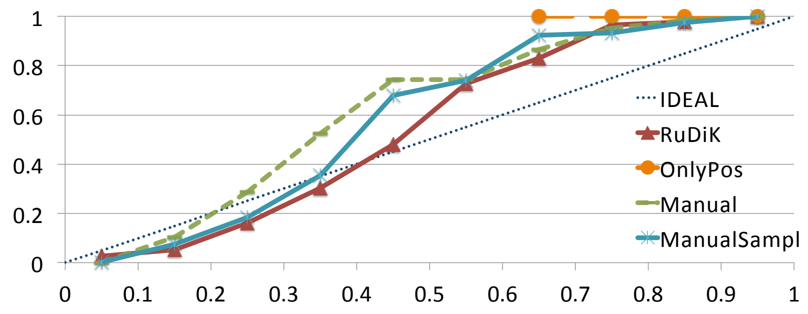


Figure 6.5: DeepDive Application 1M articles

6.4.4 Ablation Study

In this last set of experiments we measured the impact of RuDiK relevant features in order to quantify the benefit of three main aspects in rules discovery. We run RuDiK on DBPEDIA with different settings than the standard ones. We report results on the same top-5 predicates of Section 6.4.1 for both positive and negative rules.

Effect of Literals. Since previous KB rules discovery approaches exclude literals from the mining [15, 78] (TODO: cite Sigmod ontological), we wanted to quantify the impact of having literal rules. Thus we run RuDiK excluding all literal values. Table 6.8 reports the output precision with and without literals. Including literal values in the mining has a considerable impact on final accuracy, both for positive and negative rules. The effect is particularly evident for negative rules, where excluding literals involves finding less than half potential errors (numbers in brackets) with a lower precision. `founder` is the most evident example: RuDiK discovers 79 potential errors with a 95% precision with literal rules, while there are not output errors with rules without literals.

Surpisingly, including literals reduces also the running time. This is due to the pruning effect of the A^* search: if we include literals the algorithm can find rather soon valid literal rules, which causes the pruning of several paths on the graph. If we exulted literals instead these paths cannot be pruned and needs to be inspected by the algorithm, which entails a bigger search space.

	With Literals		**Without Literals**	
Type	*Run Time*	*Precision*	*Run Time*	*Precision*
Positive Rules	~35min	**63.99%**	~54min	60.49%
Negative Rules	~20min	**92.38%** (499)	~25min	84.85% (235)

Table 6.8: Rules Accuracy without Literals on DBPEDIA.

Rules Length Impact.

Table 6.9: *maxPathLen* Parameter Impact on DBPedia.

	$MaxPathLen = \mathbf{2}$		$MaxPathLen = \mathbf{3}$		$MaxPathLen = \mathbf{4}$	
Type	*Run Time*	*Precision*	*Run Time*	*Precision*	*Run Time*	*Precision*
Positive	∼**3min**	49.17%	∼35min	**63.99**%		
Negative	∼**56sec**	90% (131)	∼20min	**92.38**% (499)		

Negative Examples Generation. A key point in RuDiK is the generation of negative examples with a modified version of the LCWA (Section 6.2.1). We therefore evaluate two different strategies to generate negative examples. Given a target predicate p from a KB $\mathcal{K}$, we call t_a the most common type of entities that are subject of p, and t_b the most common type for the object. As an example, if $p =$`founder`, $t_a = Company$ and $t_b = Person$. We define k as the number of triples having p as predicate in $\mathcal{K}$ – k is the cardinality of the positive examples set. The first alternative negative examples generation strategy is *Random*: we randomly select k pairs (x, y) from the cartesian product of all entities x of type t_a and all entities y of type t_b, such that the triple $\langle x, p, y\rangle \notin \mathcal{K}$. The second generation strategy, named *LCWA_Random*, leverages on the LCWA. In *LCWA_Random* we randomly pick k pairs (x, y) from the cartesian product of all entities x of type t_a and all entities y of type t_b, such that: *(i)* $\langle x, p, y'\rangle \in \mathcal{K}$, with $y' \neq y$; *(ii)* $\langle x', p, y\rangle \in \mathcal{K}$, with $x' \neq x$; *(iii)* $\langle x, p, y\rangle \notin \mathcal{K}$. *LCWA_Random* is equivalent to RuDiK generation strategy, minus the constriction that x and y must be connected by a predicate different from p. The constraint of having k examples aims at reducing the size of the cartesian product that can easily explode otherwise.

Table 6.10 reports the accuracy of discovering negative rules with the three different generation strategies (RuDiK is our modified LCWA strategy). *Random* and *LCWA_Random* strategies show very similar behaviours, with a slightly better precision than RuDiK. This is because whenever we randomly pick examples from the cartesian product of subject and object, the likelihood of picking entities from a different time period is very high, and negative rules pivoting on time constraints are usually correct. As an example, a correct rule for the target predicate `child` is `birthDate(`a, v_0`)` $\wedge$ `birthDate(`b, v_1`)` $\wedge\, v_0 > v_1 \Rightarrow \neg$`child(`$a, b$`)`, stating that whenever

Table 6.10: Effect of Negative Examples Generation Strategy on DBPedia.

Strategy	*# Potential Errors*	*Precision*
Random	247	**95.95%**
LCWA_Random	263	95.82%
RuDiK	**499**	92.38%

a person a is born after a person b, then b cannot be child of a. The likelihood of randomly picking some pairs of persons (x, y) where x is born after y is very high, hence such kind of rules will often be discovered. However these are the *only* kind of rules that random generation strategies are able to find, as shown from the smaller number of errors that they can identify. Instead, the constriction of forcing x and y to be connected by a different predicate generates negative examples with heterogeneous properties that lead to different results. Rules such as $\texttt{parent}(a, b) \Rightarrow \neg\texttt{spouse}(a, b)$ are very unlikely to be generated with random strategies, since the likelihood of picking two people that are in a parent relation is very low. The generation strategy we adopt in RuDiK allows the discovery of more types of rules, and not only rules involving time constraints. This has multiple benefits: on the one hand, we discover more rules, which entails discovering a higher number of errors (Table 6.10); on the other hand, different rule types lead to high quality negative examples, that can be used in several applications such as training Machine Learning algorithms (see Section 6.4.3).

6.5 Related Work

Our work is inspired by Inductive Logic Programming, but uses techniques from dependencies discovery in relational databases, KBs construction, and mining graph patterns. We review a few of the most relevant works below.

Relational Database Constraints. A significant body of work has addressed the problem of discovering constraints over relational database sources. Functional Dependencies (FDs), a formalism to express relations and dependencies among attributes, has been studied in constraints discovery literature for more than 20 years [17], with a recent focus on performance [16]. FDs can be grouped into two strands: the schema-level approaches [91] (similar to our positive rules discovery), and the instance-driven approaches [145]. More recently, Conditional FDs extend standard FDs by enforcing patterns of semantically related constants [67]. In the context of inconsistencies discovery for relational data, Denial Constrains (DCs) are the current state-of-the-art techniques [39]. DCs are a universally quantified first order logic formalism to express constraints over relational data, and they are directly related to our negative output rules. Efficient DCs algorithms have been proposed for data cleaning and consistent query answering [26, 40].

Despite being directly related to our output and expressing a richer language, FDs and DCs cannot be applied to RDF databases for three main reasons: *(i)* the schema-less nature of RDF data and the closed world assumption which no longer holds on RDF KBs; *(ii)* FDs and DCs techniques rely on the assumption that data is either clean or has a negligible amount of errors; *(iii)* scalability issues on large RDF dataset: applying relational database techniques on RDF KBs would imply to materialise all possible predicates combinations into relational tables. Some FDs approaches focus on dataset with erroneous data [14, 97], however they are still inapplicable to RDF data due to scalability problems.

KBs Rule Mining. Recently, the focus for constraints and rules discovery is moving towards RDF databases. The closest works to ours are AMIE [78] and OP algorithm (TODO: cite Ontological Path Finding), which discover positive Horn Rules from RDF KBs with same language biases. They both uses the same discovery algorithm: it first loads the entire KB into memory and then, working one predicate at time, tries to expand rules by connecting a predicate to others that share common variables. Rules are ranked according to a confidence measure that leverages on KBs partial closed world assumption. Our graph generation and navigation technique is similar to their approach, however our examples generation allows us to discover rules on just a small fraction of the KB. This is beneficial not only from a scalability point of view (see Section 6.4.2), but also from the language perspective: neither of the two approaches can afford smart literals comparisons. We have not tested our running time against (TODO: cite Ontological Path Finding) because we could not find an implementation of it, however it requires a powerful cluster of several machines to split the KB into multiple nodes. We showed in the experimental section how RuDiK outperforms AMIE both in final accuracy and running time.

[15] is an instance-level approach that discovers new facts for specific entities rather than generic variables. We showed in the experimental section that RuDiK is capable of generating more facts with a better precision with just a single rule. (TODO: cite Sigmod data lakes) is a modern system to discover Conditional Denial Constraints (CDCs) from RDF Data. Differently from other systems, it includes literals in its language. CDCs can be directly mapped to our negative rules, however there is not a general correlation between CDCs and positive rules. Another major difference of our setting is the hardware: our disk-based approach is designed to handle large KBs with limited memory resources, while (TODO: cite Sigmod Lakes) works on a distributed environment with a total of 832 GB RAM memory.

To the best of our knowledge, RuDiK is the first approach that is generic enough to use the same algorithm to discover both positive and negative rules in RDF KBs.

Inductive Logic Programming. Inductive Logic Programming (ILP) is a subfield of Machine Learning and Logic Programming which investigates the inductive construction of first-order Horn Rules from examples and background knowledge, usually expressed through logic formalisms [114]. RuDiK can be seen as an ILP system where the KB is the background knowledge, and the generation and validation sets correspond to positive and negative training examples.

WARMR is an ILP system that discovers frequent patterns (expressed through DATALOG queries) that succeed with respect to a sufficient number of examples [55]. When translated to databases, such patterns correspond to conjunctive queries. ALEPH[11] is an available ILP system that is based on Prolog Inverse Entailment [113]. ALEPH works iteratively, by selecting examples from the background knowledge. It first constructs the most specific clause that entails the example selected (*bottom clause*), and then searches for some subset of the literals in the bottom clause that has the *best score* in order to define more general rules. The system allows the user to choose among several scoring functions. ILP systems such as WARMR and ALEPH are designed to work with

[11]`http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/aleph`

a closed world assumption, and always require the definition of positive and negative examples. AMIE clearly outperfroms these two systems [78], showing evident scalability issues when dealing medium-size KBs. Moreover, one of the main limitations of classic ILP systems is the assumption of having high-quality, errors-free training examples. We showed how this assumption does not longer hold on incomplete and erroneous KBs.

Sherlock [126] is an interesting ILP system that learns first-order Horn Rules from Web text. One of the key advantage of Sherlock is being unsupervised: it does not require negative training examples. It uses statistical significance and statistical relevance in order to discover rules that exceeds a given threshold. Differently from our setting, Sherlock is specifically designed to learn Horn Rules from open domain facts that are extracted from the Web. An interesting future direction is to adapt RuDiK to discover rules from relations extracted from free text rather than on a well-defined KB.

ILP systems take inspiration from Association Rule Mining [18], where given a database of costumer transactions the goal is to discover all frequent itemsets, i.e., all combinations of items that are found together with some minimum confidence. A well known example in a supermarket database could state that 90% of transactions that purchase bread and butter also purchase milk. As previously mentioned, adapting such a relational database setting to KBs would require the materialisation of all possible predicates combinations into relational tables.

Eventually, most of the ILP and KBs rules discovery systems rank rules according to a support value, and output only those rules that exceed a given threshold. We showed in Section 6.4.2 that properly setting such thresholds is not trivial, as often good rules are preceded in rank by meaningless ones. RuDiK does not use any threshold and outputs rules only when coverages over generation and validation sets are considered acceptable.

Relation to other areas. Our examples generation strategy leverages on the Local Closed World Assumption (LCWA). When dealing with incomplete KBs, the LCWA is a popular technique that replaces the canonical Closed World Assumption of standard relational databases. The LCWA has been used in Google Knowledge Vault to estimate the quality of extracted triples [59, 57], AMIE [78] uses the LCWA to penalise discovered rules, and sometimes the LCWA is used to evaluate the quality of a target KB [60]. We see our examples generation strategy as complementary to our approach. It is possible to run RuDiK with any input examples, no matter how such examples have been generated.

Our graph-based rules discovery approach is close in spirit to mining graph patterns [63, 154], where given a (big) input graph the goal is to discover the most frequent patterns (subgraphs) according to some scoring functions. Our setting presents a key difference: we primary look at edge labels and we are not interested in node labels, since nodes are mapped to variables when translating subgraphs to Horn Rules. Moreover, given the portion of the graph between two entities x and y, we are interested in discovery *all* possible subgraphs between x and y, which makes the problem easily solvable with BFS-like techniques.

Another interesting setting is the one of SpiderMine [152]. SpiderMine looks for the top-K largest patterns in a graph, where each node in the pattern is at most at distance r from a head vertex u. In our setting we do not have a single head vertex, but rather many vertexes (starting nodes) from which we begin the path computation. Furthermore our goal is not to find graph patterns (subgraphs), but we are simply interested in all possible paths between a pair of vertexes.

Fan et. al.(TODO: cite "Wenfei Fan. Functional Dependencies for Graphs") laid the theoretical foundations of Functional Dependencies on Graphs (GFDs). They also propose parallel algorithms for GFDs computation and evaluate the accuracy on Yago and DBPedia. Despite existing a natural correlation between FDs and Horn Rules, the language they propose covers only a portion of our negative rules to detect inconsistencies in graph databases. Moreover their language does not include smart literals comparisons, shown to be vital when detecting errors in KBs.

Chapter 7

Conclusion and Future Work

TODO: Conclusions.

- MUST TODO: talk about optimisation techniques to cut down running time in RuDiK

Bibliography

[1] MUC7.
`http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2001T02`.

[2] Reuters. `http://about.reuters.com/researchandstandards/corpus/index.asp`.

[3] **AlchemyAPI**. `www.alchemyapi.com`.

[4] **CiceroLite**.
`www.languagecomputer.com/products/text-annotation/cicerolite.html`.

[5] **LingPipe**. `http://alias-i.com/lingpipe/`.

[6] **Lupedia**. `www.old.ontotext.com/lupedia`.

[7] **OpenCalais**. `www.opencalais.com`.

[8] **ROSeAnn**. `http://diadem.cs.ox.ac.uk/roseann`.

[9] **Saplo**. `http://saplo.com/api`.

[10] **WADaR**. `http://diadem.cs.ox.ac.uk/wadar`.

[11] **Wikimeta**. `www.wikimeta.com`.

[12] **Yahoo Analysis API**. `https://developer.yahoo.com/contentanalysis/`.

[13] **Zemanta**. `www.zemanta.com`.

[14] Ziawasch Abedjan, Cuneyt G Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. Temporal rules discovery for web data cleaning. *Proceedings of the VLDB Endowment*, 9(4):336–347, 2015.

[15] Ziawasch Abedjan and Felix Naumann. Amending rdf entities with new facts. In *European Semantic Web Conference*, pages 131–143. Springer, 2014.

[16] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. Dfd: Efficient functional dependency discovery. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 949–958. ACM, 2014.

[17] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[18] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[19] Arnon Amir, Marco Berg, Shih-Fu Chang, Winston Hsu, Giridharan Iyengar, Ching-Yung Lin, Milind Naphade, Apostol Natsev, Chalapathy Neti, Harriet Nock, et al. Ibm research trecvid-2003 video retrieval system. *NIST TRECVID-2003*, 2003.

[20] Javed A Aslam and Mark Montague. Models for metasearch. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 276–284. ACM, 2001.

[21] Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction for the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.

[22] Roberto Basili, Alessandro Moschitti, Maria Teresa Pazienza, and Fabio Mossimo Zanzotto. Personalizing web publishing via information extraction. *IEEE Intelligent systems*, (1):62–70, 2003.

[23] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *VLDB*, volume 1, pages 119–128, 2001.

[24] Oliver Bender, Franz Josef Och, and Hermann Ney. Maximum entropy models for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 148–151. Association for Computational Linguistics, 2003.

[25] Michael Benedikt and Christoph Koch. Xpath leashed. *ACM Computing Surveys (CSUR)*, 41(1):3, 2009.

[26] Leopoldo Bertossi. Database repairing and consistent query answering. *Synthesis Lectures on Data Management*, 3(5):1–121, 2011.

[27] Leopoldo Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4):407–434, 2008.

[28] Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and dtds. *ACM Transactions on Database Systems (TODS)*, 35(2):11, 2010.

[29] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web*, 7(3):154–165, 2009.

[30] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 143–154. ACM, 2005.

[31] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[32] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and integration of partially overlapping web sources. *Proceedings of the VLDB Endowment*, 6(10):805–816, 2013.

[33] Paul Buitelaar, Philipp Cimiano, Stefania Racioppa, and Melanie Siegel. Ontology-based information extraction with soba. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2006.

[34] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

[35] Rafael Carreira, Sónia Carneiro, Rui Pereira, Miguel Rocha, Isabel Rocha, Eugénio C Ferreira, and Anália Lourenço. Semantic annotation of biological concepts interplaying microbial cellular responses. *BMC bioinformatics*, 12(1):460, 2011.

[36] Luying Chen, Stefano Ortona, Giorgio Orsi, and Michael Benedikt. Aggregating semantic annotators. *Proceedings of the VLDB Endowment*, 6(13):1486–1497, 2013.

[37] Boris Chidlovskii, Bruno Roustant, and Marc Brette. Documentum eci self-repairing wrappers: Performance analysis. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 708–717. ACM, 2006.

[38] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. Tegra: Table extraction by global record alignment. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1713–1728. ACM, 2015.

[39] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.

[40] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 458–469. IEEE, 2013.

[41] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1247–1261. ACM, 2015.

[42] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. Context-aware wrapping: synchronized data extraction. In *Proceedings of the 33rd international conference on Very large data bases*, pages 699–710. VLDB Endowment, 2007.

[43] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[44] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. Towards the self-annotating web. In *Proceedings of the 13th international conference on World Wide Web*, pages 462–471. ACM, 2004.

[45] Fabio Ciravegna and Alberto Lavelli. Learningpinocchio: Adaptive information extraction for real world applications. *Natural Language Engineering*, 10(02):145–165, 2004.

[46] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd international conference on Very large data bases*, pages 315–326. VLDB Endowment, 2007.

[47] Eli Cortez, Altigran S da Silva, Marcos André Gonçalves, and Edleno S de Moura. Ondux: on-demand unsupervised learning for information extraction. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 807–818. ACM, 2010.

[48] Eli Cortez, Daniel Oliveira, Altigran S da Silva, Edleno S de Moura, and Alberto HF Laender. Joint unsupervised structure discovery and information extraction. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 541–552. ACM, 2011.

[49] Valter Crescenzi and Giansalvatore Mecca. Automatic information extraction from large websites. *Journal of the ACM (JACM)*, 51(5):731–779, 2004.

[50] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6).* 2011.

[51] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 541–552. ACM, 2013.

[52] Nilesh Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 335–348. ACM, 2009.

[53] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230, 2011.

[54] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment*, 5(7):680–691, 2012.

[55] Luc Dehaspe and Hannu Toivonen. Discovery of frequent datalog patterns. *Data Mining and knowledge discovery*, 3(1):7–36, 1999.

[56] Omkar Deshpande, Digvijay S Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Building, maintaining, and using knowledge bases: a report from the trenches. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1209–1220. ACM, 2013.

[57] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[58] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Integrating conflicting data: the role of source dependence. *Proceedings of the VLDB Endowment*, 2(1):550–561, 2009.

[59] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *Proceedings of the VLDB Endowment*, 7(10):881–892, 2014.

[60] Xin Luna Dong, Evgeniy Gabrilovich, Kevin Murphy, Van Dang, Wilko Horn, Camillo Lugaresi, Shaohua Sun, and Wei Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *Proceedings of the VLDB Endowment*, 8(9):938–949, 2015.

[61] Deborah Duong, Jim Venuto, Ben Goertzel, Ryan Richardson, Shawn Bohner, and Edward Fox. Support vector machines to weight voters in a voting system of entity extractors. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 1226–1230. IEEE, 2006.

[62] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078–1089, 2009.

[63] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7):517–528, 2014.

[64] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Web-scale information extraction in knowitall:(preliminary results). In *Proceedings of the 13th international conference on World Wide Web*, pages 100–110. ACM, 2004.

[65] Jérôme Euzenat. Semantic precision and recall for ontology alignment evaluation. In *IJCAI*, pages 348–353, 2007.

[66] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Cleaning inconsistencies in information extraction via prioritized repairs. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 164–175. ACM, 2014.

[67] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering*, 23(5):683–698, 2011.

[68] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment*, 3(1-2):173–184, 2010.

[69] Wenfei Fan, Shuai Ma, Nan Tang, and Wenyuan Yu. Interaction between record matching and data repairing. *Journal of Data and Information Quality (JDIQ)*, 4(4):16, 2014.

[70] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[71] David Ferrucci and Adam Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

[72] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[73] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Querying and repairing inconsistent numerical databases. *ACM Transactions on Database Systems (TODS)*, 35(2):14, 2010.

[74] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 168–171. Association for Computational Linguistics, 2003.

[75] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.

[76] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Diadem: Thousands of websites to a single database. *Proceedings of the VLDB Endowment*, 7(14):1845–1856, 2014.

[77] Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, and Andrew Sellers. Oxpath: A language for scalable data extraction, automation, and crawling on the deep web. *The VLDB Journal*, 22(1):47–72, 2013.

[78] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24(6):707–730, 2015.

[79] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Ajax: an extensible data cleaning tool. In *ACM Sigmod Record*, volume 29, page 590. ACM, 2000.

[80] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. Corroborating information from disagreeing views. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 131–140. ACM, 2010.

[81] Paul Suganthan GC, Chong Sun, Haojun Zhang, Frank Yang, Narasimhan Rampalli, Shishir Prasad, Esteban Arcaute, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, et al. Why big data industrial systems need rules and what we can do about it. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 265–276. ACM, 2015.

[82] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. The llunatic data-cleaning framework. *Proceedings of the VLDB Endowment*, 6(9):625–636, 2013.

[83] Anna Lisa Gentile, Ziqi Zhang, Isabelle Augenstein, and Fabio Ciravegna. Unsupervised wrapper induction using linked data. In *Proceedings of the seventh international conference on Knowledge capture*, pages 41–48. ACM, 2013.

[84] Andrew V Goldberg and Robert E Tarjan. Efficient maximum flow algorithms. *Communications of the ACM*, 57(8):82–89, 2014.

[85] Davide Grossi and Gabriella Pigozzi. Introduction to judgment aggregation. In *Lectures on Logic and Computation*, pages 160–209. Springer, 2012.

[86] Frank E Grubbs. An introduction to probability theory and its applications. *Technometrics*, 9(2):342–342, 1967.

[87] Dan Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of mathematical biology*, 55(1):141–154, 1993.

[88] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 775–784. ACM, 2011.

[89] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[90] Stephan Hartmann, Gabriella Pigozzi, and Jan Sprenger. Reliable methods of judgement aggregation. *Journal of Logic and Computation*, 20(2):603–617, 2010.

[91] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

[92] Sham Kakade, Yee Whye Teh, and Sam T Roweis. An alternate objective function for markovian fields. In *ICML*, pages 275–282, 2002.

[93] Nanda Kambhatla. Minority vote: at-least-n voting improves recall for extracting relations. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 460–466. Association for Computational Linguistics, 2006.

[94] Anitha Kannan, Inmar E Givoni, Rakesh Agrawal, and Ariel Fuxman. Matching unstructured product offers to structured product specifications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 404–412. ACM, 2011.

[95] Vangelis Karkaletsis, Constantine D Spyropoulos, Claire Grover, M Pazienza, Jose Coch, and Dimitris Souflis. A platform for cross-lingual, domain and user adaptive web information extraction. In *ECAI*, volume 16, page 725, 2004.

[96] Atanas Kiryakov, Borislav Popov, Ivan Terziev, Dimitar Manov, and Damyan Ognyanoff. Semantic annotation, indexing, and retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):49–79, 2004.

[97] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.

[98] Judice LY Koh, Mong Li Lee, Wynne Hsu, and Kai Tak Lam. Correlation-based detection of attribute outliers. In *Advances in Databases: Concepts, Systems and Applications*, pages 164–175. Springer, 2007.

[99] Zornitsa Kozareva, Óscar Ferrández, Andrés Montoyo, Rafael Muñoz, Armando Suárez, and Jaime Gómez. Combining data-driven systems for improving named entity recognition. *Data & Knowledge Engineering*, 61(3):449–466, 2007.

[100] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[101] Kristina Lerman, Steven Minton, and Craig A Knoblock. Wrapper maintenance: A machine learning approach. *J. Artif. Intell. Res.(JAIR)*, 18:149–181, 2003.

[102] Liping Ma and John Shepherd. Information extraction using two-phase pattern discovery. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 534–535. ACM, 2004.

[103] Ashwin Machanavajjhala, Arun Shankar Iyer, Philip Bohannon, and Srujana Merugu. Collective extraction from heterogeneous web lists. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 445–454. ACM, 2011.

[104] Alexander Maedche, Günter Neumann, and Steffen Staab. Bootstrapping an ontology-based information extraction system. In *Intelligent exploration of the web*, pages 345–359. Springer, 2003.

[105] Imran R Mansuri and Sunita Sarawagi. Integrating unstructured data into relational databases. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 29–29. IEEE, 2006.

[106] MarketLine. Online retail in the united states, 2013. On-line at `http://www.marketresearch.com/MarketLine-v3883/Online-Retail-United-States-7760207/`.

[107] Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. Maximum entropy markov models for information extraction and segmentation. In *ICML*, volume 17, pages 591–598, 2000.

[108] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8. ACM, 2011.

[109] Xiaofeng Meng, Dongdong Hu, and Chen Li. Schema-guided wrapper maintenance for web-data extraction. In *Proceedings of the 5th ACM international workshop on Web information and data management*, pages 1–8. ACM, 2003.

[110] Eirinaios Michelakis, Rajasekar Krishnamurthy, Peter J Haas, and Shivakumar Vaithyanathan. Uncertainty management in rule-based information extraction systems. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 101–114. ACM, 2009.

[111] Matthew Michelson and Craig A Knoblock. Creating relational data from unstructured and ungrammatical data sources. *Journal of Artificial Intelligence Research*, pages 543–590, 2008.

[112] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, pages 777–782, 2013.

[113] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[114] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[115] Aditya Parameswaran, Nilesh Dalvi, Hector Garcia-Molina, and Rajeev Rastogi. Optimal schemes for robust web extraction. In *Proceedings of the VLDB Conference*, volume 4. VLDB Endowment, 2011.

[116] Aditya Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-assisted graph search: it's okay to ask questions. *Proceedings of the VLDB Endowment*, 4(5):267–278, 2011.

[117] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[118] Vijayshankar Raman and Joseph M Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[119] Juan Raposo, Alberto Pan, Manuel Álvarez, and Justo Hidalgo. Automatically maintaining wrappers for semi-structured web sources. *Data & Knowledge Engineering*, 61(2):331–358, 2007.

[120] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

[121] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

[122] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[123] Giuseppe Rizzo and Raphaël Troncy. Nerd: a framework for unifying named entity recognition and disambiguation extraction tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–76. Association for Computational Linguistics, 2012.

[124] Riccardo Rosati. On the complexity of dealing with inconsistency in description logic ontologies. *Trans (R)*, 1:C2, 2011.

[125] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008.

[126] Stefan Schoenmackers, Oren Etzioni, Daniel S Weld, and Jesse Davis. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1088–1098. Association for Computational Linguistics, 2010.

[127] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceedings of the 10th ACM workshop on Web information and data management*, pages 9–16. ACM, 2008.

[128] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment*, 8(11):1310–1321, 2015.

[129] Luo Si, Tapas Kanungo, and Xiangji Huang. Boosting performance of bio-entity recognition by combining results from multiple systems. In *Proceedings of the 5th international workshop on Bioinformatics*, pages 76–83. ACM, 2005.

[130] Shaoxu Song, Hong Cheng, Jeffrey Xu Yu, and Lei Chen. Repairing vertex labels under neighborhood constraints. *Proceedings of the VLDB Endowment*, 7(11):987–998, 2014.

[131] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble learning for named entity recognition. In *The Semantic Web–ISWC 2014*, pages 519–534. Springer, 2014.

[132] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706. ACM, 2007.

[133] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[134] Fabian M Suchanek, Mauro Sozio, and Gerhard Weikum. Sofie: a self-organizing framework for information extraction. In *Proceedings of the 18th international conference on World wide web*, pages 631–640. ACM, 2009.

[135] Marieke Van Erp, Giuseppe Rizzo, and Raphaël Troncy. Learning with the web: Spotting named entities on the intersection of nerd and machine learning. In *# MSM*, pages 27–30. Citeseer, 2013.

[136] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, 4(9):528–538, 2011.

[137] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[138] Daisy Zhe Wang, Michael J Franklin, Minos Garofalakis, Joseph M Hellerstein, and Michael L Wick. Hybrid in-database inference for declarative information extraction. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 517–528. ACM, 2011.

[139] Haochang Wang and Tiejun Zhao. Identifying named entities in biomedical text based on stacked generalization. In *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pages 160–164. IEEE, 2008.

[140] Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 457–468. ACM, 2014.

[141] Michael Wick, Aron Culotta, and Andrew McCallum. Learning field compatibilities to extract database records from unstructured text. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 603–611. Association for Computational Linguistics, 2006.

[142] Michael L Wick, Khashayar Rohanimanesh, Karl Schultz, and Andrew McCallum. A unified approach for schema matching, coreference and canonicalization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 722–730. ACM, 2008.

[143] Tak-Lam Wong, Wai Lam, and Tik-Shun Wong. An unsupervised framework for extracting and normalizing product attributes from multiple web sites. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 35–42. ACM, 2008.

[144] Dekai Wu, Grace Ngai, and Marine Carpuat. A stacked, voted, stacked model for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 200–203. Association for Computational Linguistics, 2003.

[145] Catharine Wyss, Chris Giannella, and Edward Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 101–110. Springer, 2001.

[146] Mohamed Yakout, Laure Berti-Équille, and Ahmed K Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 553–564. ACM, 2013.

[147] Yanhong Zhai and Bing Liu. Extracting web data using instance-based learning. In *Web Information Systems Engineering–WISE 2005*, pages 318–331. Springer, 2005.

[148] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, pages 76–85. ACM, 2005.

[149] Chen Jason Zhang, Lei Chen, Yongxin Tong, and Zheng Liu. Cleaning uncertain data with a noisy crowd. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 6–17. IEEE, 2015.

[150] Chang Zhao, Jalal Mahmud, and IV Ramakrishnan. Exploiting structured reference data for unsupervised text segmentation with conditional random fields. In *SDM*, pages 420–431. SIAM, 2008.

[151] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th international conference on World Wide Web*, pages 66–75. ACM, 2005.

[152] Feida Zhu, Qiang Qu, David Lo, Xifeng Yan, Jiawei Han, and Philip S Yu. Mining top-k large structural patterns in a massive network. *Proceedings of the VLDB Endowment*, 4(11):807–818, 2011.

[153] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, 2004.

[154] Lei Zou, Lei Chen, and M Tamer Özsu. Distance-join: Pattern match query in a large graph database. *Proceedings of the VLDB Endowment*, 2(1):886–897, 2009.