Efficient Pruning of Large Knowledge Graphs

Stefano Faralli[1*], **Irene Finocchi**[2], **Simone Paolo Ponzetto**[3] and **Paola Velardi**[2]
[1] University of Rome Unitelma Sapienza
[2] University of Rome Sapienza
[3] University of Mannheim
stefano.faralli@unitelma.it, {finocchi,velardi}@di.uniroma1.it, simone@informatik.uni-mannheim.de

Abstract

In this paper we present an efficient and highly accurate algorithm to prune noisy or over-ambiguous knowledge graphs given as input an extensional definition of a domain of interest, namely as a set of instances or concepts. Our method climbs the graph in a bottom-up fashion, iteratively layering the graph and pruning nodes and edges in each layer while not compromising the connectivity of the set of input nodes. Iterative layering and protection of pre-defined nodes allow to extract semantically coherent DAG structures from noisy or over-ambiguous cyclic graphs, without loss of information and without incurring in computational bottlenecks, which are the main problem of state-of-the-art methods for cleaning large, i.e., Web-scale, knowledge graphs. We apply our algorithm to the tasks of pruning automatically acquired taxonomies using benchmarking data from a SemEval evaluation exercise, as well as the extraction of a domain-adapted taxonomy from the Wikipedia category hierarchy. The results show the superiority of our approach over state-of-art algorithms in terms of both output quality and computational efficiency.

1 Introduction

In the age of information, the Web provides a goldmine of data from which knowledge can be harvested on an unprecedented scale. As a matter of fact, efforts in information extraction and knowledge acquisition from the past decade have been able to produce knowledge resources on a scale that was arguably hard to imagine a few years ago [Carlson *et al.*, 2010; Wu *et al.*, 2012; Fader *et al.*, 2011; Gupta *et al.*, 2014; Dong *et al.*, 2014, *inter alia*]. Large coverage, however, comes with new challenges associated with the noise in the input data, as well as errors in the output knowledge base.

In this paper, we address the problem of pruning large knowledge graphs, removing "noisy" edges and relations. We specifically focus on a high-performing, yet efficient algorithm since in this task we are typically faced with complexity issues that arise from the large size of the graph to be pruned. Examples include open-domain Web mining [Seitner *et al.*, 2016] or pruning large crowdsourced knowledge bases – e.g., those considered in [Faralli *et al.*, 2015a] and [Kapanipathi *et al.*, 2014]. We present a new algorithm, named CRUMB-TRAIL, to efficiently and effectively mine a taxonomic structure "hidden" within a large graph, starting from a number of constraints (or "crumbs", as in the fairy tale of Hansel and Gretel) that a user can select to characterize a domain of interest, which help identifying promising paths.

[*]Contributions made while he was still at the University of Mannheim.

2 Problem Statement

We start by formally defining the task of pruning a knowledge graph, along with an intuition and real-case examples to justify the utility of such a task. We define a **knowledge graph** as a *typed directed graph* $KG = (V, E, T)$: nodes V identify a set of concepts or entities and E is a set of relations across nodes such that $(a, b, t) \in E$, where $a, b \in V$ and $t \in T$ is the relation type. Relations types may vary due to the model specification and the scopes of the representation.

Definition 1 (Strict weak order relation)**.** *A relation $\prec$ of type t is a strict weak order (SWO) if it is irreflexive, antisymmetric, and transitive [Roberts and Tesman, 2009].*

The edges of a knowledge graph KG induce a strict weak order if the edges in the transitive closure of KG satisfy the three properties of Definition 1. In this paper, we restrict to the case in which all edges are of a given type t, where t is a SWO relation. Common types of SWO relations in a knowledge graph are hypernymy, meronymy and topical [Ponzetto and Strube, 2011] relations.

Given a knowledge graph KG, we identify two *undesired phenomena*, which we denote as "noise":

i) *infringements of SWO relations:*

if, for some relation, at least one of the three conditions in Definition 1 does not hold; ii) *unessential nodes and edges:* assuming that KG is the graph representation of a specific knowledge domain (like, for example, health, tourism, or architecture), unessential nodes and edges are those which are either redundant or even harmful to describe the domain semantics, for instance due to their ambiguity and the resulting potential for semantic shifts. Accordingly, we define a **noisy knowledge graph** NKG as a graph where one or both undesired phenomena occur.

Pruning a noisy knowledge graph NKG is viewed throughout the paper as the process of searching a *subgraph* of NKG such that: 1) its edges induce a strict weak order and 2) it does not contain unessential nodes and edges. *Condition 1* is satisfied if the subgraph does not contain cycles: in fact, for each pair of nodes u and v in a cycle, both $u \prec v$ and $v \prec u$ hold by transitivity ($u \prec v$ means that edge (u, v) exists), since the two nodes can reach each other along cycle edges, violating the property of being antisymmetric. This can be formally verified using only topological properties of the graph. On the contrary, testing *condition 2* should in principle involve the use of knowledge-based methods, which are heuristic in nature. Our challenge in this paper is to adopt a notion of (un)essentiality which can be formally tested on the graph topology without resorting to methods for assessing loosely defined constraints such as "domain semantics". To this end, our approach is to assume the availability of an initial set of nodes P, defined as follows:

Definition 2 (Primitive essential nodes)**.** *Given a noisy knowledge graph $NKG = (V, E)$, let $P \subseteq V$ be a set of primitive essential nodes containing:*

a) *terminological nodes:* a subset of nodes of NKG which belong to a target domain of interest (i.e., domain-specific instances and terms);

a) *categorical nodes (the "crumbs"):* some (even few) domain-pertinent concepts of NKG expected to be located in "higher" positions in the strict weak order;

c) *root:* a concept r that is a common ancestor of all nodes in P, i.e., a node from which all nodes of P can be reached (in short, we call this property r*-connectivity*).

Given P, we characterize *essential nodes* in terms of connectivity of P with respect to the root r:

Definition 3 (Essential nodes)**.** *Essential nodes in a NKG are those nodes that are strictly needed to $preserve$ set P during the pruning process, i.e., those nodes which, if removed, would compromise r-connectivity.*

The identification of (un)essential nodes is at the core of the CRUMBTRAIL algorithm. The intuition is that, provided that set P implicitly defines one or even more domains of interest, noisy nodes and edges – either originated by extraction errors or out-of-domain – should mostly lie *outside* the relevant paths connecting terminological and crumb nodes to the root r, i.e., they are not essential.

Further note that the output subgraph *is different from the subgraph of NKG induced by P*: in particular, it might contain nodes that are not in P. Similarly, while resembling the maximal acyclic subgraph [Berger and Shor, 1990] and the minimum feedback arc set [Demetrescu and Finocchi, 2003] problems, which are often used for untangling the structure of complex networks [Sugiyama *et al.*, 1981; Demetrescu and Finocchi, 2001], computing such subgraphs would not be appropriate for our pruning task, since connectivity properties would not be necessarily preserved. On the other side directed Steiner tree [Charikar *et al.*, 1999] algorithms, besides computational complexity issues (see Sections 3 and 5), would maintain connectivity but would prune NKG very aggressively, removing all multiple paths from r to any node in P, at the risk of omitting nodes that could be semantically relevant to describe domain-consistent and equally valid classifications of a concept.

Examples of noise. In NKG, the presence of noise might be the result of automatic [Velardi *et al.*, 2013; Mitchell *et al.*, 2015] or collaborative [Ponzetto and Strube, 2011] knowledge graph construction. In fact, it is acknowledged that the process of creating large and dynamically updated knowledge resources [Hoffart *et al.*, 2016] cannot be entrusted to a small team of domain experts, and is therefore subject to errors. Some common examples of "noise", which illustrate the utility of NKG cleaning, include:

- **Infringement of SWO relations:** cycles, which represent an infringement of SWO relations, are very common in the Wikipedia category graph[1] (e.g., $Persian\_books \rightarrow Iranian\_books \rightarrow Persian\_books$) and in dictionaries. Circularity of definitions is a well-known issue in lexicography and is considered to be a problem since the earlier history of computational linguistics [Richens, 2008].
- **Extraction errors:** the problem of automated taxonomy learning is commonly addressed by extracting SWO relations such as hypernymy relations from glossaries [Navigli and Velardi, 2010] or definitional patterns [Hearst, 1992] like: "x are y" ("cats are felines"). Although more or less sophisticated, all algorithms are prone to extraction errors. As an example, the sentence "cats are examples of highly refined adaptation and evolution" may lead to extracting the following hypernymy relations: $cats \leftarrow example$, $cats \leftarrow refined$, $cats \leftarrow adaptation$, which are all wrong according to commonsense knowledge.
- **Out-of-domain nodes:** in Wikipedia, Freebase, DBpedia and other large knowledge bases, categorical information is freely generated by contributors with limited editorial verification, which leads to an excessive multiple inheritance, a problem that may cancel the advantage of adding semantics [Matykiewicz and Duch, 2014]. A typical example from Wikipedia's category graph is shown in Table 1: note that two very different entities, a director (David Lynch) and an education institution (University of Tokyo), end up almost in the same set of upper Wikipedia categories (oddly, they both reach the categories $Education$ and $People$). The problem however is not so much the quality of semantic relations, but rather the coexistence of many different perspectives. For example, the page *University of Tokyo* is classified (see column 2 of Table 1) as *National universities* and *Visitor attractions in Tokyo*. These are both reasonable classifications, however the first would be an appropriate category for an *Education* taxonomy, and the second for a *Tourism and places* taxonomy. Furthermore, these different "semantic threads" do not remain separated while climbing towards upper categories, but they interweave over and over again in the category graph, making the (useful) task of generating domain views particularly complex.

[1] https://en.wikipedia.org/wiki/Wikipedia:Dump_reports/Category_cycles

Wikipage: David Lynch

1st level categories	2nd level categories	Top categories
American people of Finnish descent, American male voice actors, Surrealist filmmakers, American television directors, **(more…)**	Filmmakers from California, Experimental filmmakers by nationality, Male voice actors by nationality, **(more…)**	Geography, People, Humans, World, History, Information, Education, Knowledge, Arts, Industry, Language, **(more…)**

Wikipage: University of Tokyo

1st level categories	2nd level categories	Top categories
Educational institutions established in 1877, 1964 Summer Olympic venues, Universities and colleges in Tokyo, **(more…)**	Sport in Japan by sport, Visitor attractions in Tokyo, College athletics conferences in Japan, National universities, **(more…)**	Geography, Humans, Science, History, Knowledge, People, Industry, Education, Technology, Employment, **(more…)**

Table 1: Example of multiple inheritance in the Wikipedia category graph: note that top categories are almost completely overlapping.

Our hunch is that both extraction errors and out-of-domain nodes are expected to be unessential for preserving the connectivity of the set P. For example, if we aim at building an animal taxonomy, with crumbs such as *mammal* and/or *animal*, and a text mining algorithm extracts multiple hypernymy relations (some of which are either wrong or out-of-domain) such as $cats \leftarrow feline$, $cats \leftarrow example$, $cats \leftarrow musical$, it is very unlikely that the nodes $example$ and $musical$ lie on hypernymy chains connecting cat with $mammal$ or $animal$, and even more unlikely that they are essential to preserve the connectivity between these nodes.

3 Related Work

Approaches to knowledge graphs cleaning in literature differ both in the method to identify relevant and irrelevant nodes, and in the higher or lower impact of the pruning policy.

1) Domain-aware "soft" pruning. Most taxonomy pruning approaches require that the user selects the relevant concepts by hand [Swartout, 1996] or describes in some way the domain of interest [Best and Lebiere, 2010]. These approaches are rather conservative, and the number of deleted nodes is actually quite low [Kim *et al.*, 2007]. Furthermore, none of these methods is able to detect and remove cycles, and complexity may also be an issue, since it is necessary to compute all the paths top-down from root to leaf nodes.

2) Domain-aware "aggressive" pruning. A number of papers rely on more "aggressive" pruning policies based on topological graph pruning methods. [Kozareva and Hovy, 2010] propose an algorithm to induce a taxonomy from a graph structure. It uses a root term, a set of seed examples of hypernymy relations (e.g., $cats \leftarrow feline$) and lexico-syntactic patterns to learn automatically from the Web hyponyms and hypernyms subordinated to the root. It then uses cycle pruning and "longest path" heuristics to induce a DAG structure. Similarly, In Ontolearn Reloaded [Velardi *et al.*, 2013] the algorithm starts from a set of automatically extracted terms and iteratively extracts hypernym relations thus building an hypernymy graph. To induce a taxonomy from the graph, the authors use a variant of Chu-Liu Edmonds' (CLE) optimal branching algorithm [Edmonds, 1967], in which the node weighting strategy is based on preserving both longest paths and the highest coverage of input terms.

3) Domain-unaware "soft" pruning approaches. Studies like [Kapanipathi *et al.*, 2014] and [Faralli *et al.*, 2015a] look at the problem of removing cycles in Wikipedia in a user recommendation task. The first paper uses simulated annealing to identify relevant upper categories starting from a set of Wikipages representing users' interests. The latter uses an efficient variant of Tarjan's topological sorting [Tarjan, 1972] for cycle pruning. To avoid a random selection of edges to prune, [Sun *et al.*, 2017] combine different heuristics to approximate an "aggressive" node ranking function and experiment different strategies to select an edge to be removed and break cycles.

All previous methods, setting aside the precision of the pruning process, present at least one of the following two problems, if not both: 1) **Computational complexity:** Some of the above approaches rely on time or space expensive techniques. For example, the complexity of CLE, and other Steiner algorithms, is affected by the need to compute the weight of alternative branches, which in general implies a depth first search (DFS). Similar problems arise in soft pruning methods, since they need to compute all paths from root to leaf nodes using DFS. Parallelization techniques such as MapReduce would not help and would rather be harmful, due to the parallelization overhead, since DFS is inherently sequential [Reif, 1985]; 2) **Information loss:** Edge or path pruning based either on topological (nodes outdegree, longest or shortest paths, transitive closure, etc.) or random selection, may cause the loss of possibly relevant hierarchical relations (especially if the shortest path heuristics is adopted, like in [Kapanipathi *et al.*, 2014]), and even the disconnection of selected seed nodes [Faralli *et al.*, 2015a].

In marked contrast, as shown in this paper, our CRUMBTRAIL algorithm does not incur in space and time limits (even when pruning extremely large and dense graphs such as the full Wikipedia), and is both domain aware and "aggressive", while preserving all the available information on the domain (the set P).

4 The CrumbTrail Algorithm

In this section we summarize our pruning algorithm, called CRUMBTRAIL. In line with the problem description provided in Section 1, given a (directed) noisy knowledge graph $NKG(V, E)$ (hereafter denoted G for brevity), a set $P \subseteq V$ of terminological nodes and crumbs to be preserved (referred to as *protected* nodes), and a root $r \in P$, CRUMBTRAIL prunes G to obtain an *acyclic* subgraph G_P that contains all nodes of P, as well as possibly other nodes to guarantee connectivity properties as explained below. *Unessential* nodes, namely redundant nodes that hinder exposing the domain-focused structure due to the presence of multiple alternative paths (Section 1), are eliminated by CRUMBTRAIL, resulting in a more aggressive pruning. The output graph G_P is *layered*, with top-down edges connecting upper to lower layers. Nodes of P can appear *on any layer*, including intermediate ones: this is in line with the fact that P contains both terminological nodes and crumbs. Moreover, under the assumption

Algorithm 1: CRUMBTRAIL

Input: preprocessed graph $G(V, E)$, $P \subseteq V$, root $r \in P$
Output: acyclic and r-connected graph G_P
1 Initialize sets $Ground$, $Intermediate$, and $postponed$ table
2 $\ell = 0$
3 $V_0 = Ground$
4 **repeat**
5 let $P_\ell = V_\ell \cap P$
6 $Ground = Ground \cup P_\ell$
7 $Intermediate = Intermediate \setminus P_\ell$
8 $\ell = \ell + 1$
9 `createNewLayer`(G, $Ground$, $Intermediate$, r, $V_{\ell-1}$, $postponed$)
10 `postponeNodes`(G,V_ℓ, $postponed$)
11 `pruneUnessential`(G, V_ℓ, $Ground$, $Intermediate$, r)
12 **until** $r \in Ground$ **and** $postponed$ *is empty*

that G is r-connected with respect to P (i.e., contains directed paths from the root r to every other node of P), CRUMBTRAIL also *preserves the r-connectivity* in G_P.

Overview. The algorithm is called after the fairy tale of Hansel and Gretel: intuitively, it finds its path towards home – the root r – following a trail of "breadcrumbs" – the set P of protected nodes. Differently from previous approaches, CRUMBTRAIL climbs G *bottom-up*, rather than top-down, traversing crumbs from a bottom layer V_0 containing a subset of P up to a layer V_h containing the root r. The number h of layers of G_P is not known *a priori*. Instead, due to the presence of cycles and multiple paths between nodes, layers V_i of G_P are *unfolded incrementally* while climbing G upwards, according to the following criteria: a) no path in G_P can connect any pair of nodes in V_i. If such a path is found during the construction of V_i, its start node is deferred (*postponed*) to an upper layer; b) cycles involving edges incident to nodes of V_i are broken, taking care of eliminating cycle edges whose removal does not disconnect any of the nodes P from the root r; c) for each node in the new layer V_i, incoming edges start from unprocessed or postponed nodes and outgoing edges end in lower layers (i.e., layers V_t such that $t < i$); d) nodes in V_i (and their incident edges) are pruned whenever unessential to preserve the r-connectivity.

Traversing and pruning G bottom-up guarantees that *the number of alternative paths decreases progressively as the graph is incrementally unfolded*, due to criteria b and d. As demonstrated in Section 5, this results in lower space consumption and faster execution with respect to previous approaches, even when processing large and highly connected graphs.

Preprocessing. As a preliminary step, we eliminate from G all self loops, all edges leading to the root (if any), and all nodes of $V \setminus P$ with indegree or outdegree equal to 0. For each node in P, we also break cycles passing through its outgoing edges. None of these operations harms the connectivity between r and nodes in P.

Data structures. CRUMBTRAIL maintains a layer counter $\ell \geq 0$ and a hash table of postponed nodes consisting of pairs $\langle v, i \rangle$, where i is an estimate of the layer at which node v will be analyzed and is used as key in the dictionary. With a slight abuse of notation, throughout this section we denote by $postponed(i)$ the subset of nodes temporarily postponed to layer i. Throughout the execution of CRUMBTRAIL, P is partitioned into two sets, called $Ground$ and $Intermediate = P \setminus Ground$. At the beginning, $Ground$ contains only P nodes with outdegree 0. The remaining P nodes are added to $Intermediate$ as well as to the $postponed$ table, using as a tentative layer the length of a shortest path to a ground node. In subsequent iterations, P nodes not yet analyzed are transferred from $Intermediate$ to $Ground$ as they are processed and assigned to layers. The first layer, V_0, coincides with $Ground$ nodes (line 3 of Algorithm 1).

Main iteration. After data structure initialization (lines 1–3 of Algorithm 1), graph pruning is performed by an iterative procedure (lines 4–12) that repeats the following steps, until the root has been visited and the postponed table is empty (termination condition at line 12):

1) `createNewLayer`: build a new layer V_ℓ and remove cycles passing through edges outgoing from nodes of V_ℓ (line 9); 2) `postponeNodes`: postpone nodes of the current layer V_ℓ that reach each other (line 10). Namely, if two nodes of V_ℓ are connected by a path of length k, the starting node of the path is temporarily postponed to layer $V_{\ell+k}$; 3) `pruneUnessential`: remove unessential nodes in V_ℓ (line 11) while preserving the r-connectivity with respect to nodes in P.

In more details, the three subroutines work as follows.

Algorithm CreateNewLayer. Besides selecting candidate nodes to be added to the current layer V_ℓ, this subroutine also removes cycles passing through incident edges. Nodes added to V_ℓ can be either starting nodes of edges whose target is in $V_{\ell-1}$ or postponed nodes.

First, every node u with an outgoing edge to $V_{\ell-1}$ is added to V_ℓ, provided that there is no directed path from u to any postponed node p. This check is done to satisfy the node layering criteria: all edges should flow top-down, but adding u to V_ℓ would create at least a bottom-up edge if u is connected to a postponed node. It may be the case that after this phase the current layer V_ℓ remains empty. If this is the case, and if there are no nodes postponed to layer ℓ, the algorithm skips to the first non-empty layer.

Next, `createNewLayer` removes cycles passing through edges outgoing from nodes in V_ℓ. The cycle breaking procedure iterates over these edges and detects a cycle across an edge (x, y) whenever it finds a path $\pi = y \rightsquigarrow x$ starting at node y and ending in x. To implement this check, the algorithm first computes a BFS tree T_r rooted at r. Notice that, if an edge f does not belong to T_r, it can be safely removed from the cycle involving (x, y) without compromising r-connectivity. Moreover, since T_r is acyclic, such an edge must necessarily exist in cycle $\langle x, y \rightsquigarrow x \rangle$ identified by the cycle breaking procedure. Hence, every cycle can be safely broken while maintaining r-connectivity.

We remark that, after the execution of this subroutine, it may be possible that there are still edges between nodes in V_ℓ: if such an edge (x, y) exists, we are guaranteed that it is not part of a cycle, and x will be postponed to an upper layer immediately later by subroutine `postponeNodes`. It may be also the case that there are edges outgoing from V_ℓ and

reaching nodes that are not yet assigned to a layer: if such an edge (x, y) exists, with $x \in V_\ell$ and y unlayered, it can happen neither that y is postponed nor that there is a path from y to a postponed node. In that case, x would have not been added to V_ℓ. Since postponed nodes include intermediate nodes (see the data structure description), if upward edges exist starting from V_ℓ, they must lead to nodes that are unessential to preserve r-connectivity and can be later removed by subroutine `pruneUnessential`.

Algorithm PostponeNodes. This subroutine is invoked by CRUMBTRAIL immediately after building a new layer and identifies nodes of V_ℓ to be postponed to subsequent layers, updating the $postponed$ table. As previously observed, CRUMBTRAIL aims at creating layers so that edges flow top-down. Hence, nodes of the current layer V_ℓ who are the starting points of paths ending in V_ℓ itself must be shifted to higher layers. In more details, given two nodes u and v in V_ℓ connected by a path $\pi = u \rightsquigarrow v$, u is postponed $|\pi|$ levels higher than v, where $|\pi|$ denotes the path length. Since there could be multiple paths connecting u and v, we do not attempt at establishing *a priori* the exact layer distance among the two nodes. Instead, π is chosen arbitrarily and the layer at which u is postponed might be later increased due to the discovery of additional (longer) paths originating from u. The $postponed$ table is updated accordingly and all the rescheduled nodes are removed from V_ℓ.

Algorithm PruneUnessential. Nodes that are not essential to preserve the connectivity between intermediate and ground nodes can be removed by the procedure `pruneUnessential`. This algorithm finds nodes of the current layer either reachable from intermediate nodes or reaching ground nodes. Nodes of V_ℓ that cannot be removed unless compromising the connectivity between $Intermediate$ and $Ground$ are called *essential*. For any $v \in V_\ell$, let $\mathcal{G}(v)$ be the set of ground nodes reachable from v and let $\mathcal{I}(v)$ be the set of intermediate nodes from which v is reachable. A node $v \in V_\ell$ is *essential* if there is at least one pair of nodes, $x \in \mathcal{I}(v)$ and $y \in \mathcal{G}(v)$, that can be connected *only* through v: i.e., x and y would be disconnected by deleting v. It follows that, if $\mathcal{G}(v) = \emptyset$ or $\mathcal{I}(v) = \emptyset$, then v does not connect any intermediate-ground pair and can be safely removed. Unessential nodes are then ranked based on $|\mathcal{G}(v)|$ and $|\mathcal{I}(v)|$ and the highest ranked node is removed (ties are broken arbitrarily). Since the removal of any node changes the rank of the others, the procedure is iterated until no more nodes can be deleted. Hence, only essential nodes survive in V_ℓ. As a last step, the algorithm deletes nodes with indegree or outdegree equal to 0 that might have been created during the previous steps.

Example. Figure 1 presents a step-by-step execution of an iteration of CRUMBTRAIL on a 21-node noisy graph. As shown in Figure 1.i, $P = \{r, a, b, c, d, e\}$. During preprocessing (Figure 1.ii), self-loops on nodes 3 and 9, as well as nodes 10, 11, and 12, whose in/out-degree is 0, are removed. Edge $(6, r)$ points to the root and is thus deleted. Edges $(2, e)$ and $(d, 1)$ are eliminated in order to break cycles $\langle e, 4, 2, e\rangle$ and $\langle d, 1, d\rangle$, respectively. The protected nodes e and r are marked as intermediate and postponed to upper layers, based on the length of a shortest path to ground nodes $\{a, b, c, d\}$. In Figure 1.iii, `createNewLayer` builds a new layer V_1 containing all nodes with an outgoing edge to V_0. The algorithm then breaks three cycles $\langle 7, 6, 8, 7\rangle$, $\langle 6, 8, 6\rangle$, and $\langle 14, 4, 14\rangle$ by removing edges $(7, 6)$, $(6, 8)$, and $(14, 4)$, respectively. Notice that after edge deletion the connectivity between intermediate and ground nodes is still maintained. In Figure 1.iv, `postponeNodes` moves node 4 out of V_1, deferring its processing to layer V_2 due to the existence of paths $\langle 4, 5, b\rangle$ and $\langle 4, 2, c\rangle$. Nodes 7, 6, 5, 2, and 14 are unessential to preserve the connectivity of r and e with ground nodes, and are thus eliminated by `pruneUnessential` in Figure 1.v. After the first iteration, only nodes 1 and 15 remain in V_1, as shown in Figure 1.vi.

5 Evaluation

In order to benchmark CRUMBTRAIL (hereafter, *CRU*), we consider the following competing approaches:

1) the algorithm from [Faralli *et al.*, 2015a] (*TAR*), which is based on Tarjan's topological sorting [Tarjan, 1972]; 2) the algorithm from [Velardi *et al.*, 2013] (*CLE*), which is based on Chu-Liu/Edmond's optimal branching [Edmonds, 1967] and an *ad-hoc* node weighting policy; 3) the Wikipedia hierarchical edge pruning algorithm proposed in [Kapanipathi *et al.*, 2014] (*HPA*).

To the best of our knowledge, these three algorithms are the only ones that tackle the problem of fully automated pruning of large Web-size knowledge graphs. As summarized in Section 3: *TAR* and *HPA* are "soft" methods only aimed at eliminating cycles, while *CLE* and *CRU* belong to the category of "aggressive" pruning algorithms. Furthermore, both *TAR* and *CLE* are lossy, i.e., they are not guaranteed to preserve the connectivity of nodes in P. *HPA* is not lossy with respect to leaf nodes, but given the simple pruning strategy, it may end up eliminating relevant edges.

Experimental setting. We evaluate the performance of the aforementioned methods for two different tasks, namely *ontology pruning* and *ontology domain adaptation*. We are given a noisy knowledge graph $NKG(V, E)$ (G for brevity) a *gold-standard* graph $GS(V', E')$ and a *reference graph* $R(V'', E'') = G(V, E) \cap GS(V', E')$. In ontology learning $G(V, E)$ is an automatically learned structure that may have noisy and missing elements with respect to the gold standard. Instead, in ontology adaptation $G(V, E)$ is a possibly very dense knowledge graph, and the aim is to extract a domain taxonomy which is fully embedded in G.

Performance metrics. We apply our four pruning algorithms to $G(V, E)$ and obtain a pruned graph $G_p(V_p, E_p)$. Importantly, for the sake of comparison with *CRU* (see line 3 of Algorithm 1), we remove from the pruned graphs obtained from each of the compared algorithms the "pending" leaf and root nodes. We recall that pending leaf nodes in a noisy graph are those leaf nodes not in P and pending roots are those different from r. As a consequence, even though *HPA* is not lossy – since its simple edge pruning strategy guarantees that all nodes $v \in V$ are preserved – in our implementation a number of peripheral nodes are eventually pruned.

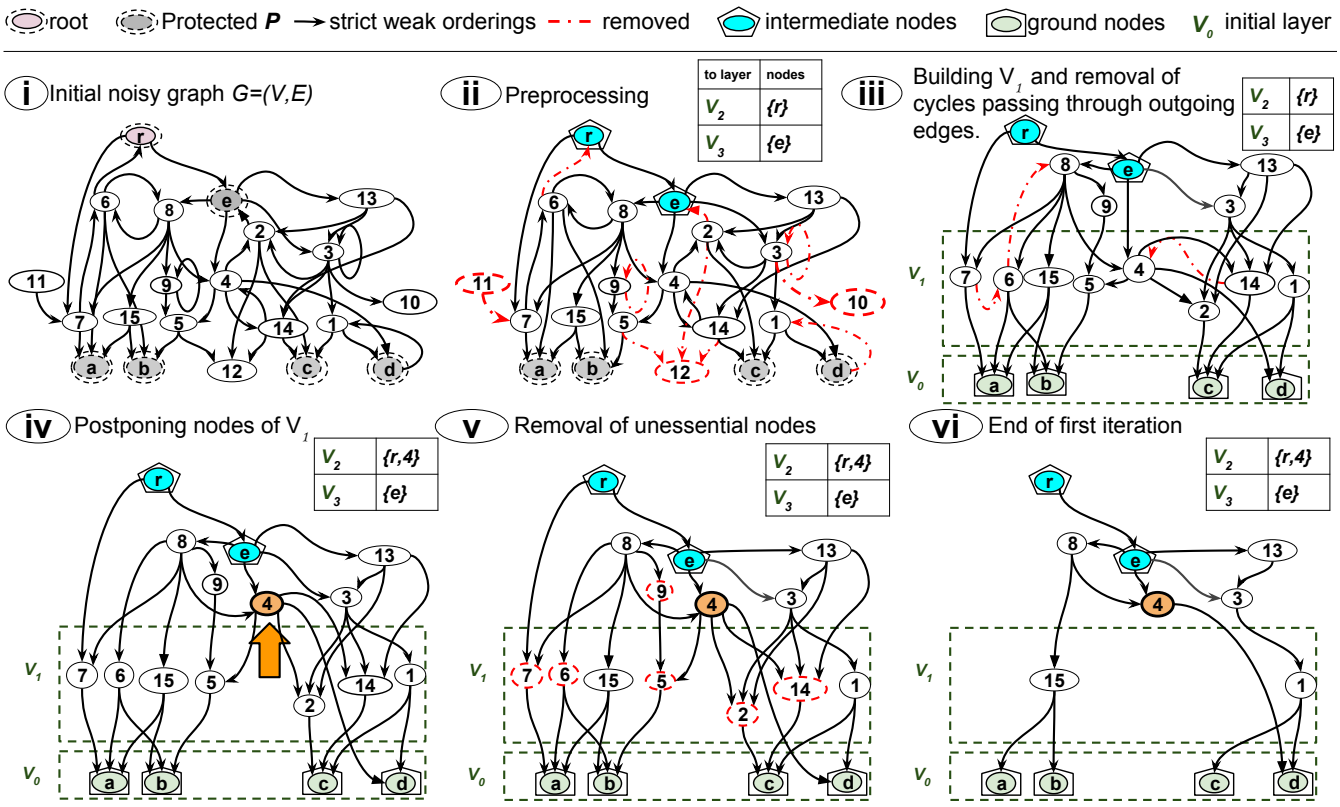

Figure 1: A complete walkthrough example of the application of CRUMBTRAIL to a noisy graph.

Ideally, we would like to have the pruned graph to perfectly match the reference graph, namely $G_p(V_p, E_p) = R(V'', E'')$ or to be able to quantify different degrees of similarity between the two graphs when no perfect match is attained. To this end, we use the *Jaccard distance* over the two node sets as evaluation measure:

$$J_V = \frac{|V'' \cup V_p| - |V'' \cap V_p|}{|V'' \cup V_p|} \quad (1)$$

Since in both tasks of ontology pruning and domain adaptation, the algorithms are provided with a number of initial nodes P, a second objective is that the ontology pruning task is not lossy, i.e. that all nodes in P are preserved in G_p. Accordingly, we compute the *coverage* of P nodes as: $C_P = \frac{|P \cap V_p|}{|P|}$. Next, in order to provide a measure of the difficulty of the pruning task, we define the following indexes to compute the amount of noise of $G(V, E)$ with respect to the reference graph $R(V'', E'')$: $Noise_V = (1 - \frac{|V''|}{|V|})$, $Noise_E = (1 - \frac{|E''|}{|E|})$. Finally, we compute the familiar metrics of precision, recall and balanced F-measure of the node P_V, R_V and $F1_V$ and edge P_E, R_E and $F1_E$ pruning tasks.

Experiment 1: Cleaning automatically learned taxonomies. In our first experiment, we use the gold-standard taxonomies and the taxonomy learning systems from Task 17 of the SemEval 2015 challenge[2]. In this shared task, competing systems were required to learn a taxonomic structure given an input terminology T and a root node r ($P = T \cup r$). Four domains were considered, namely *Chemical*, *Food*, *Equipment* and *Science*. For each domain, the participants were asked to automatically induce their own taxonomies, using the terminology P as the leaf nodes of the taxonomy. The participating systems thus output automatically built hypernymy graphs, with some amount of noisy and missing nodes and edges with respect to the four gold-standard taxonomies.

We apply the four previously listed pruning algorithms (*HPA*, *TAR*, *CLE* and *CRU*) to the output of each of the ontology learning systems participating in the SemEval 2015 challenge, giving a total of 31 runs (note that not all systems submitted an NKG for each of the 4 domains). Table 2 provides an overview of the characteristics of the data used in our

Taxonomy type	Avg. # nodes	Avg. # edges
Gold (GS)	557	634
Reference (R)	558	225
SemEval submitted runs ($NKGs$)	754	1385
average $Noise_V$	0.26%	
average $Noise_E$	0.84%	
Pruning algorithm	Avg. # nodes	Avg. # edges
HPA	289	739
TAR⋆⋆	288	764
CLE⋆	263	500
CRU	299	823

Table 2: Structural analysis of the dataset used for ontology pruning.

[2] http://alt.qcri.org/semeval2015/task17/

	nodes					edges		
	C_P	J_V	P_V	R_V	$F1_V$	P_E	R_E	$F1_E$
HPA	**1.0**	.24	.90	**.98**	.93	.23	**.94**	.34
TAR	.81	.35	.76	.81	.77	.18	.80	.28
CLE	.72	.42	.75	.73	.73	.22	.63	.29
CRU	**1.0**	**.21**	**.97**	.94	**.95**	**.24**	.93	**.35**

Table 3: Performance in the ontology pruning task (best results for each evaluation metric are bolded).

first experiment, covering structural properties of the gold-standard, reference noisy (i.e., SemEval submitted runs) and pruned taxonomies. All the data shown in the table have been averaged over each type of taxonomy, considering for each submitted NKG only the main connected component (the one which contains the root node). Note that, as mentioned in the table, in some cases *TAR* and *CLE* could not guarantee the connectivity of any of the nodes in P. Also note that, as shown by the average size of the reference graph, in the average, $NKGs$ submitted to the SemEval challenge include the majority of nodes in the gold taxonomy ($V'' \approx V'$) but they are unable to capture many hypenymy relations ($E'' \ll E'$). Table 3 shows the performance of the different pruning algorithms. In the SemEval challenge, the submitted noisy hypernymy graphs are not very large (cf. Table 2): consequently none of the algorithms incurs in complexity problems. However, both *TAR* and *CLE* are lossy – i.e., some of the nodes in P are disconnected from the resulting taxonomy (cf. the coverage of P nodes C_P in column 2). In particular, *TAR* and *CLE* are lossy in 11 and 19 out of 31 runs, respectively. Concerning the quality of the resulting pruned taxonomies, the best performance figures are obtained by *CRU* and *HPA*. *HPA* shows a higher recall, primarily due to the fact that it performs only cycle pruning. *CRU* instead achieves the best overall results when using precision, F_1 measure and the Jaccard distance as evaluation metric – in the case of Jaccard, the smaller the value, the better the taxonomy, since we measure here the distance of the pruned taxonomy from the gold-standard one.

We note that all approaches perform worst on edge pruning than on node pruning. This is because reference taxonomies R do not cover many of the edges of the gold-standard taxonomies in the first place, as shown in Table 2. That is, since pruning algorithms cannot, and are not meant to be able to retrieve missing nodes and edges in $GS \setminus R$, limited coverage of the edges of the gold-standard taxonomy GS in the reference taxonomy R heavily impacts overall performance on edge structuring. Missing edges are also the main cause for pruning preserved nodes in lossy algorithms (i.e., *TAR* and *CLE*). In Table 3, for the sake of fair comparison, we do not test the unique feature of CRUMBTRAIL, which is able to identify promising paths in the noisy graph, when given a number of intermediate nodes as additional hints (the "crumbs"). However, we found that, when adding to the set P an increasing but small number of "crumbs" randomly selected from the intermediate nodes of the reference taxonomies R, performance indicators that were already high achieve only very small improvements, whereas performance on edge pruning, which was lower, increases of about 30% before saturating.

Experiment 2: Domain adaptation of the Wikipedia category hierarchy. In our second experiment, we apply the four pruning algorithms to the entire Wikipedia category graph, thus testing the full power of the CRUMBTRAIL algorithm. For this, we use different subgraphs of the Wikipedia category graph as silver-standard datasets, namely the category hierarchies rooted in the categories *Singers*, *Entertainers* and *Companies*. The silver-standard category hierarchies are obtained as follows: 1) starting from the full Wikipedia category graph, we remove all incoming edges in the selected root node (e.g., for *Singers*, we remove edges starting in *Singing*, *Vocal Ensembles* and *Musicians*); 2) we compute the transitive closure of the root node r, e.g., $Closure$(*Singers*); 3) we add to the gold-standard all the nodes in $Closure(r)$ and all edges (v_i, v_j) such that $v_i, v_j \in Closure(r)$.

Note that this approach is not guaranteed to produce an error-free, gold-standard category hierarchy. For example, back to Table 1, selecting the root node *Geography*, we would oddly reach both *David Lynch* and *University of Tokyo*. However, for very focused intermediate concepts such as those selected in this experiment, we manually verified on large samples of the datasets that nodes are indeed mostly 'golden' (e.g., they can be considered as specializations of the three roots according to commonsense). The four compared algorithms are provided with: i) the set $P = T \cup r$, where T are the Wikipages at the leaf nodes of $Closure(r)$ (e.g., *Diamanda Galás* under category *Singers*), and ii) the full Wikipedia category graph rooted in *Main topic classifications*. The task for each algorithm is then to induce from the "noisy" Wikipedia category graph a domain-focused hierarchy, i.e., a directed acyclic graph "embedded" in it, with root r and leaf nodes T. The result of each algorithm is then compared with the silver standard, which, in contrast to the previous experiment, is completely included in the Wikipedia category graph.

Table 5 compares the four algorithms and shows a striking superiority of *CRU* over the other methods, both in terms of Jaccard distance and F-measure, for all domains. Though the Jaccard distance is always remarkably low, the relative rankings (not shown for sake of space) reflect the order of generality of the domain, and thus, the impact of multiple inheritance: from the most focused domain *Singers*, to the most generic *Entertainers*. Table 4 shows that the amount of noise in this second experiment is much higher than in the previous experiment using SemEval, and that the dimensions of both noisy and silver taxonomies are order of magnitude higher. The Table also shows that *CRU* and *CLE* (the latter, whenever it does not run out of memory) are much more aggressive in pruning nodes and edges, as expected, given that *HPA* and *TAR* are "soft" algorithms. In terms of *efficiency CRU* is always faster, followed by the naive *HPA* pruning strategy. Instead, for *CLE* it was necessary to limit the maximum depth h of depth-first search to 5 (denoted as $CLE(5)$ in Table 4), and even with this limit, a solution was actually produced only for the first domain, namely *Singers*. In the other two cases, *CLE* runs out of memory on a multi-core machine. *CRU* is beaten in terms of recall by some of the other methods, since, as we repeatedly remarked, *TAR* and *HPA* only remove cycles. Note that, in terms of Jaccard distance, *CRU* almost perfectly retrieves the silver category hierarchies from the noisy

Wikipedia category graph (G)	# nodes	# edges	
root:Main topic classifications	5,398,723	18,006,438	
Silver category hierarchy ($GS = R$)	# nodes	# edges	
root: Singers	49,076	93,243	
$Noise_V$	99.09%		
$Noise_E$	99.48%		
algorithms	# nodes	# edges	run time
HPA	167,268	588,681	0.5 h
TAR	167,279	589,564	6 h
$CLE(5)$	48,317	84,945	5 h
CRU	48,632	91,686	0.1 h
Silver category hierarchy ($GS = R$)	# nodes	# edges	
root: Entertainers	338,853	844,708	
$Noise_V$	93.72%		
$Noise_E$	95.30%		
algorithms	# nodes	# edges	run time
HPA	529,839	1,859,316	1 h
TAR	538,682	2,099,215	1.5 days
$CLE(5)$	n.a	n.a	2 days (fail)
CRU	324,576	774,508	0.2 h
Silver category hierarchy ($GS = R$)	# nodes	# edges	
root: Companies by stock exchange	7,953	9,436	
$Noise_V$	99.85%		
$Noise_E$	99.94%		
algorithms	# nodes	# edges	run time
HPA	58,543	177,279	1 h
TAR	54,312	159,432	1.5 days
$CLE(5)$	n.a	n.a	2 days (fail)
CRU	7,934	9,395	0.14 h

Table 4: Structural analysis of the Wikipedia category graphs.

	nodes					edges		
	C_P	J_V	P_V	R_V	$F1_V$	P_E	R_E	$F1_E$
HPA	**1.0**	.65	.35	.99	.48	.19	.90	.28
TAR	.99	.53	.35	**.99**	.40	.61	**.99**	.31
CLE_5	.33	.67	.33	.33	.33	**.33**	.30	.32
CRU	**1.0**	**.02**	**1.0**	.98	**.99**	**1.0**	.98	**.96**

Table 5: Performance in the task of domain adaptation of the Wikipedia category graph. Measures are averaged on the 3 domains.

graphs. The only competitive system in terms of quality of the pruned graph is *CLE*: however, computational complexity prevents from obtaining a solution as the dimension of the taxonomy increases, a problem that cannot be mitigated with parallelization algorithms. The higher performance obtained by *CRU* in the Wikipedia experiment is also due to the fact that, in contrast to the SemEval experiment, the silver category graph is fully embedded in the NKG: therefore, all the necessary information is available to the pruning algorithm. Further note that in this second experiment we did not test the additional feature of CRUMBTRAIL of expanding the set P with intermediate categories: however, the performances are already extremely good and there is quite a limited space for improvements.

To summarize, our results indicate that, as the dimension and connectivity of the NKG and the amount of noise increase, so does the superiority of CRUMBTRAIL over the other graph pruning methods, both in terms of quality of the results and speed of execution.

6 Conclusion and Future Work

To the best of our knowledge, CRUMBTRAIL is the first algorithm that has been shown to perform well in the task of removing multiple inheritance in the Wikipedia graph. This problem has prevented so far from fully exploiting Wikipedia hierarchical relations in many relevant applications – including user recommendation [Kapanipathi *et al.*, 2014; Faralli *et al.*, 2015b; Elgohary *et al.*, 2011], document categorization [Gabrilovich and Markovitch, 2006] or query understanding [Schuhmacher *et al.*, 2015], to name a few.

We also acknowledge some limits, which we aim to address in the future. First, the pruning strategy is limited to knowledge graphs with a single relation type that satisfies the SWO constraint. Although SWO relations have a predominant role, this does not fit to more general cases that might occur in the Web of Data, in which more relation types are available. Moreover, CRUMBTRAIL relies on a number terminological and categorical nodes (the"crumbs") that a user can select to characterize a domain of interest. Since the availability of categorical nodes is a tighter constraint, a better solution could be to automatically infer seed categories from terminological nodes.

Acknowledgements

This work has been partially supported by grant PO 1900/1-1 of the Deutsche Forschungsgemeinschaft (DFG) under the JOIN-T project, by the IBM Faculty Award #2305895190, and by the MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science of Sapienza University.

References

[Berger and Shor, 1990] Bonnie Berger and Peter W. Shor. Approximation algorithms for the maximum acyclic subgraph problem. In *Proc. of SODA*, pages 236–243, 1990.

[Best and Lebiere, 2010] Bradley J. Best and Christian Lebiere. Extracting the ontological structure of opencyc for reuse and portability of cognitive models. In *Proc. of BRiMS*, pages 282–286, 2010.

[Carlson *et al.*, 2010] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proc. of AAAI*, pages 1306–1313, 2010.

[Charikar *et al.*, 1999] Moses Charikar, Chandra Chekuri, To yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Liy. Approximation algorithms for directed Steiner problems. *Journal of Algorithms*, 33(1):73 – 91, 1999.

[Demetrescu and Finocchi, 2001] Camil Demetrescu and Irene Finocchi. Breaking cycles for minimizing crossings. *ACM Journal on Experimental Algorithmics*, 6:1 – 39, 2001.

[Demetrescu and Finocchi, 2003] Camil Demetrescu and Irene Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, 86(3):129–136, 2003.

[Dong *et al.*, 2014] 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 *Proc. of KDD*, pages 601–610, 2014.

[Edmonds, 1967] Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[Elgohary *et al.*, 2011] Ahmed Elgohary, Hussein Nomir, Ibrahim Sabek, Mohamed Samir, Moustafa Badawy, and Noha A. Yousri. Wiki-rec: A semantic-based recommendation system using Wikipedia as an ontology. In *Proc. of ISDA*, pages 1465–1470, 2011.

[Fader *et al.*, 2011] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proc. of. EMNLP*, pages 1535–1545, 2011.

[Faralli *et al.*, 2015a] Stefano Faralli, Giovanni Stilo, and Paola Velardi. Large scale homophily analysis in Twitter using a twixonomy. In *Proc. of IJCAI*, pages 2334–2340, 2015.

[Faralli *et al.*, 2015b] Stefano Faralli, Giovanni Stilo, and Paola Velardi. Recommendation of microblog users based on hierarchical interest profiles. *Social Network Analysis Mining*, 5(1):25:1–25:23, 2015.

[Gabrilovich and Markovitch, 2006] Evgeniy Gabrilovich and Shaul Markovitch. Overcoming the brittleness bottleneck using Wikipedia: Enhancing text categorization with encyclopedic knowledge. In *Proc. of AAAI*, pages 1301–1306, 2006.

[Gupta *et al.*, 2014] Rahul Gupta, Alon Halevy, Xuezhi Wang, Steven Whang, and Fei Wu. Biperpedia: An ontology for search applications. In *Proc. of PVLDB*, pages 505–516, 2014.

[Hearst, 1992] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proc. of COLING*, pages 539–545, 1992.

[Hoffart *et al.*, 2016] Johannes Hoffart, Dragan Milchevski, Gerhard Weikum, Avishek Anand, and Jaspreet Singh. The knowledge awakens: Keeping knowledge bases fresh with emerging entities. In *Proc. of WWW*, pages 203–206, 2016.

[Kapanipathi *et al.*, 2014] Pavan Kapanipathi, Prateek Jain, Chitra Venkataramani, and Amit Sheth. User interests identification on Twitter using a hierarchical knowledge base. In *The Semantic Web: Trends and Challenges*, volume 8465, pages 99–113. Springer, 2014.

[Kim *et al.*, 2007] Jong W. Kim, Jordi C. Caralt, and Julia K. Hilliard. Pruning bio-ontologies. In *Proc. of HICSS*, 2007.

[Kozareva and Hovy, 2010] Zornitsa Kozareva and Eduard Hovy. A semi-supervised method to learn and construct taxonomies using the web. In *Proc. of EMNLP-10*, pages 1110–1118, 2010.

[Matykiewicz and Duch, 2014] Pawel Matykiewicz and Wlodzislaw Duch. Multiple inheritance problem in semantic spreading activation networks. In *Proc. of BIH*, pages 252–265, 2014.

[Mitchell *et al.*, 2015] Tom M Mitchell, William W Cohen, Estevam R Hruschka Jr, Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, et al. Never ending learning. In *Proc. of AAAI*, pages 2302–2310, 2015.

[Navigli and Velardi, 2010] Roberto Navigli and Paola Velardi. Learning word-class lattices for definition and hypernym extraction. In *Proc. of ACL*, pages 1318–1327, 2010.

[Ponzetto and Strube, 2011] Simone Paolo Ponzetto and Michael Strube. Taxonomy induction based on a collaboratively built knowledge repository. *Artificial Intelligence*, 175(9-10):1737–1756, 2011.

[Reif, 1985] John H Reif. Depth-first search is inherently sequential. *Information Processing Letters*, 20(5):229–234, 1985.

[Richens, 2008] Tom Richens. Anomalies in the WordNet verb hierarchy. In *Proc. of COLING*, pages 729–736, 2008.

[Roberts and Tesman, 2009] Fred Roberts and Barry Tesman. *Applied combinatorics*. CRC Press, 2009.

[Schuhmacher *et al.*, 2015] Michael Schuhmacher, Laura Dietz, and Simone Paolo Ponzetto. Ranking entities for web queries through text and knowledge. In *Proc. of CIKM*, pages 1461–1470, 2015.

[Seitner *et al.*, 2016] Julian Seitner, Christian Bizer, Kai Eckert, Stefano Faralli, Robert Meusel, Heiko Paulheim, and Simone Ponzetto. A large database of hypernymy relations extracted from the web. In *Proc. of LREC*, 2016.

[Sugiyama *et al.*, 1981] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

[Sun *et al.*, 2017] Jiankai Sun, Deepak Ajwani, Patrick K. Nicholson, Alessandra Sala, and Srinivasan Parthasarathy. Breaking cycles in noisy hierarchies. In *Proc. of WebSci*, pages 151–160, 2017.

[Swartout, 1996] William R. Swartout. Toward distributed use of large-scale ontologies. In *Proc. of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996.

[Tarjan, 1972] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. on Comput.*, 1:146–160, 1972.

[Velardi *et al.*, 2013] Paola Velardi, Stefano Faralli, and Roberto Navigli. OntoLearn Reloaded: A graph-based algorithm for taxonomy induction. *Computational Linguistics*, 39(3):665–707, 2013.

[Wu *et al.*, 2012] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proc. of SIGMOD*, pages 481–492, 2012.