HKU CS Tech Report TR-2018-04

# Towards Practical Open Knowledge Base Canonicalization

Tien-Hsuan Wu[†] Zhiyong Wu[†] Ben Kao[†] Pengcheng Yin[§]
†Department of Computer Science, The University of Hong Kong
§Language Technologies Institute, Carnegie Mellon University
{thwu,zywu,kao}@cs.hku.hk,pcyin@cs.cmu.edu

## ABSTRACT

An Open Information Extraction (OIE) system processes textual data to extract *assertions*, which are structured data typically represented in the form of ⟨*subject*; *relation*; *object*⟩ triples. An Open Knowledge Base (OKB) is a collection of such assertions. We study the problem of *canonicalizing* an OKB, which is defined as the problem of mapping each *name* (a textual term such as "the rockies", "colorado rockies") to a canonical form (such as "rockies"). Galárraga et al. [18] proposed a hierarchical agglomerative clustering algorithm using canopy clustering to tackle the canonicalization problem. The algorithm was shown to be very effective. However, it is not efficient enough to practically handle large OKBs due to the large number of similarity score computations. We propose the FAC algorithm for solving the canonicalization problem. FAC employs *pruning techniques* to avoid unnecessary similarity computations, and *bounding techniques* to efficiently approximate and identify small similarities. In our experiments, FAC registers orders-of-magnitude speedups over other approaches.

## KEYWORDS

Entity Resolution; Open Knowledge Base; Canonicalization

## 1 INTRODUCTION

A *knowledge base* (KB) stores factual information about the world on which inference engines are applied to perform logical deduction to answer user questions. The Web, with its wide information coverage, is an important source for KB construction. In recent years, development in *open information extraction* (OIE) techniques has allowed tremendous amounts of web documents to be efficiently and effectively processed to build open knowledge bases (OKBs, [12, 15]). Specifically, an OIE system, such as TextRunner [9], ReVerb [14], OLLIE [8] and ClausIE [2], parses sentences in web documents to extract relational tuples, typically in the form of ⟨*subject*; *relation*; *object*⟩ triples. We call such triples *assertions*. For example, ClueWeb09[1] is a collection of 500 million English web pages. When ReVerb is applied to ClueWeb09, around 6 billion assertions are extracted, among which

$A_1$ : ⟨Barack Hussein Obama; grew up in; Honolulu⟩

$A_2$ : ⟨Barack Obama; is the president-elect of; the United States⟩

are two examples. Each field of an assertion is given by a phrase called a *surface name* (or just *name* for short), which is composed of one or more *words*. For example, the subject field of the assertion $A_1$ above is the three-word name "Barack Hussein Obama". An OKB is a collection of assertions.

To effectively use an OKB as a repository of information for logical reasoning, one needs to identify the physical entity to which a name refers. To this end, entity resolution (ER) is an important step that aims at determining if names (possibly expressed with different words) given in different assertions are referring to the same entity [1, 11]. For example, an entity resolver may determine that the subject names of assertions $A_1$ and $A_2$, even though they are different strings, refer to the same entity (president Obama). Such a resolution would allow one to deduce, for example, the fact: "the president-elect of the United States grew up in Honolulu."

One approach to solving the ER problem is *entity linking* (EL) [5, 10, 17, 20, 23, 24], which is assisted by a *curated KB*. A curated KB is a high-precision KB that is manually created. Examples include Freebase [6], DBpedia [13] and YAGO [25]. Given a name, the idea is to identify an entity in a curated KB to which the name should be linked to. Different names are considered to refer to the same entity if they share the same link target. A big disadvantage of the EL approach is that curated KBs generally have limited scopes. Hence, entity linking often fails for tail or emerging entities whose information is not covered by a curated KB. In fact, it is reported in [20] that names from one-third of the assertions extracted by ReVerb from ClueWeb09 cannot be linked to Wikipedia entities.

In [7, 18], Galárraga et al. propose a canonicalization approach to solve the ER problem. They propose an algorithm (which we will call GHMS to name it after the authors) that applies canopy clustering and hierarchical agglomerative clustering (HAC) to cluster assertions. Subject names of assertions that are grouped into the same cluster are given a canonical form and are considered to refer to the same entity. It is shown that GHMS is highly effective.

A major drawback of GHMS is its high computational cost. Since it is based on HAC, a similarity matrix, which is quadratic in size with respect to the number of assertions, has to be computed. In [18], the effectiveness of GHMS was demonstrated by applying it to canonicalize a collection of 35,630 assertions (let us call this *the 36K set*). We installed GHMS and repeated their experiment on the 36K set. The algorithm took 41 minutes to finish. Note that the 36K set is a very small set. In practice, an OKB would be much larger. For example, 6 billion assertions are derived from the 500M web pages in the ClueWeb09 collection. Even with a very aggressive scheme to filter the 6B assertions to obtain the highest-quality assertions, 2.6M assertions remain [18]. We applied GHMS to canonicalize these 2.6M ReVerb-extracted assertions. The program did not finish after 10 days on a machine with an Intel i7-6850K CPU and 128GB RAM.

[1] http://www.lemurproject.org/clueweb09.php/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*CIKM '18, October 22–26, 2018, Torino, Italy*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271707

We found that only about 36% of assertions had been processed by GHMS during that period. With less aggressive filtering of the 6B assertions, a set of 15M assertions are collected in [14]. This set is about 6 times as large as the 2.6M-assertion set and thus would have a bigger coverage of knowledge. Canonicalizing the 15M set, however, would be even more expensive; A very rough projection indicates that GHMS would probably take more than 100 days to complete, which is impractical.

The objective of this paper is to carry out an in-depth analysis of GHMS and to propose optimization strategies. The goal is to achieve canonicalization on *practically-large* OKBs within a reasonable amount of time. We put forward the FAC algorithm, which is a highly optimized version of GHMS. In our experiment, FAC took only 4.4 minutes to canonicalize the 15M-assertion set. This is orders-of-magnitude faster compared with GHMS.

We remark that an efficient solution to OKB canonicalization is highly desirable. It enables effective entity resolution, which in turn, leads to effective logical deduction using an OKB. Moreover, canonicalization helps identify and remove redundant assertions, which reduces storage requirement and speeds up query processing. As web pages are collected in mass and new names are created, timely canonicalization is essential to keep an OKB up-to-date.

Our major contributions are:

- We evaluate GHMS to identify its computation bottlenecks.
- We apply an inverted index and propose highly effective pruning techniques to avoid unnecessary similarity score computation.
- We propose bounding techniques to efficiently compute bounds of similarity scores. These bounds allow small similarities to be quickly filtered without computing their values exactly.
- We conduct extensive experiments to evaluate our algorithm FAC over a number of real datasets. Our results show that FAC is highly scalable and can be used to canonicalize practically-large OKBs.

The rest of the paper is organized as follows. Section 2 summarizes some related works. In Section 3 we formally define the canonicalization problem and describe the GHMS algorithm. In Section 4 we present our optimization techniques and the FAC algorithm. In Section 5 we present our experiment results. Finally, Section 6 concludes the paper.

## 2 RELATED WORKS

In this section we briefly describe some representative works on *entity linking*, *assertion canonicalization* and *canopy clustering* that are related to our problem and solution.

**[Entity Linking]** The entity linking (EL) approach to solving the entity resolution (ER) problem is to link each name (in an assertion) to an entity of a curated KB. For a survey on EL, see [5, 24]. Some representative works include [10, 20, 23].

The methods proposed in [10, 23] use Wikipedia as a reference curated KB and attempt to link each name to a Wikipedia entity (a wiki-page). Both methods share a select-and-rank framework — Given a name, candidate wiki-pages are first selected. Then, the candidates are ranked based on their relevancies to the name. Finally, the name is linked to the highest-ranked candidate. The methods differ in their selection criteria and ranking strategies. Specifically, in [10], candidate selection is based on string similarity (e.g., between the name and a wiki-page's title) with considerations of acronyms and aliases. Candidate ranking is based of a set of candidate wiki-page features, such as an *authority score* (the number of hyperlinks that point to the page), and a *popularity score* (the rank of the page in a google search), etc.

In [23], candidate selection is based on the *anchor text* of hyperlinks, which is the clickable, displayed text of a link. Specifically, a page $p$ is selected as a candidate for a given name $n$ if $n$ occurs frequently in the anchor text of the hyperlinks that point to page $p$. Candidate pages are then ranked based on a set of *local features* and a set of *global features*. Local features are those that measure the relatedness of the page to the name and the document from which the name is extracted. Global features, on the other hand, concern the relatedness among the candidate pages.

**[Canonicalization by Clustering]** Another approach to solving the ER problem is by canonicalization, which can be done by clustering names in assertions. Example systems include ConceptResolver [19], Resolver [26], and GHMS [18].

ConceptResolver processes names extracted by the NELL system [3]. NELL is an OIE system that extracts knowledge by reading the Web. Each name extracted by NELL is assigned a category, such as it being a "city" or a "company". HAC clustering is then applied to cluster names under each category. Names of the same cluster are considered to refer to the same entity.

Resolver clusters names given in the assertions extracted by the OIE system TextRunner. Given two subject names $s_1$ and $s_2$, if they are contained in two assertions that share the same relation and object, the two names are said to have a shared property. The similarity between $s_1$ and $s_2$ is measured by the number of shared properties and their string similarity. Resolver applies HAC to cluster subject names using the above similarity measure. Pruning techniques are applied to improve the efficiency of HAC clustering.

Our work is based on [18], in which canopy-based HAC is performed to cluster assertions and to canonicalize names. We use GHMS to refer to the algorithm given in [18]. Details of GHMS will be given in Section 3.

**[Canopy Clustering]** HAC is computationally expensive for large datasets because it requires a large number of similarity computations. To improve efficiency, McCallum, et al. propose canopy clustering [4]. The idea is to employ a *loose distant threshold* and a *fast approximate distant metric* to quickly group objects in a dataset into canopies. Each object can be a member of multiple canopies. After canopies are created, a more expensive clustering algorithm, such as HAC is applied to each canopy. Since a canopy is typically much smaller than the original dataset, clustering within a canopy is much faster than clustering the whole dataset in one shot. Token blocking, proposed by Papadakis, et al. [22], applies the idea of canopy clustering to text processing and entity resolution. To cluster a set of string objects, each string $s$ is *tokenized* into the words that are contained in $s$. A canopy (called a block in [22]) $C_w$ is created for each word $w$ such that all strings containing $w$ are collected into $C_w$. In [16], Fisher et al. propose to control the block size to improve clustering efficiency — blocks of small sizes are merged, while large blocks are shrunk.

## 3 OPEN KB CANONICALIZATION

In this section we formally define the OKB canonicalization problem. We also describe the GHMS algorithm given in [18].

### 3.1 Problem Definition

An assertion is a triple of the form ⟨*subject*; *relation*; *object*⟩, where each field is represented by a phrase, called a *name*. A name that appears in the subject field is called a *subject name*. We assume that each assertion is extracted from an identifiable document. To simplify our discussion, we assume that documents are obtained from the web and each is associated with a url (serving as a document id). Given an assertion $a = \langle s; r; o \rangle$ and the url $u$ of the document from which $a$ is extracted, we extend $a$ by $u$ to obtain a quadruple $\langle s; r; o; u \rangle$. We call such a quadruple an *assertion occurrence* of $a$. Assertion occurrences that share the same subject name and the same url are collected into a structure called a *mention*. A mention has the form ⟪*subject*; *url*; *argument*⟫, where *argument* is a list of (relation; object) pairs. For example, the two occurrences $\langle s; r_1; o_1; u \rangle$ and $\langle s; r_2; o_2; u \rangle$ share the same subject $s$ and url $u$. They are grouped into the mention $\langle\langle s; u; Arg \rangle\rangle$ where $Arg$ is a list containing $(r_1; o_1)$ and $(r_2; o_2)$. Figure 1(a) shows some example assertion occurrences and Figure 1(b) shows the mentions constructed from them. We write $x \in m$ iff an assertion occurrence $x$ is grouped into a mention $m$. Following [18], we assume that all occurrences of a subject name in the same document (url) refer to the same physical entity. Hence, all assertion occurrences that are collected into the same mention are assumed to refer to the same subject entity.

Definition 1 (OKB Canonicalization). *Given an OKB $\mathcal{X}$ of assertion occurrences, the problem of OKB canonicalization is to derive a mapping $f$ such that given any two assertion occurrences $x_1, x_2 \in \mathcal{X}$ whose subject names are $s_1$ and $s_2$, respectively, we have $f(x_1) = f(x_2)$ if and only if $s_1$ and $s_2$ represent the same entity.*

One can interpret $f(x_1)$ and $f(x_2)$ as canonical forms of the subject names $s_1$ and $s_2$, respectively. We consider two subject names the same entity if and only if they share the same canonical form. The problem is to find a function $f()$ that returns such a canonical form.

Since we assume that all assertion occurrences that are grouped into the same mention have the same subject entity, the subject names of these occurrences should be given the same canonical form. In other words, given a mention $m$, we have $f(x_1) = f(x_2)\ \forall x_1, x_2 \in m$. As a result, we only need to assign a canonical form to one assertion occurrence per mention; other occurrences in the same mention would adopt the same form.

Note that in our definition we only canonicalize subject names. This is done to simplify our discussion and to follow the approach given in [18]. This simplification also makes our algorithm FAC consistent with GHMS in terms of their output. The techniques can be extended to canonicalize object names as well.

### 3.2 GHMS

GHMS canonicalizes an OKB by clustering. The idea is to group assertion occurrences into clusters. Occurrences that share the same cluster are assumed to refer to the same subject entity. As we have discussed, assertion occurrences of the same mention are given the same canonical form. GHMS, therefore, clusters mentions instead of assertion occurrences so as to reduce the number of objects to be clustered to improve efficiency. Figure 1 illustrates the various steps of GHMS.

GHMS starts with a pre-precessing step (①) that performs two tasks: First, it transforms a collection of assertion occurrences into mentions. Second, for any word $w$ that appears in some subject names, the number of times that $w$ appears in the OKB is computed. This frequency is denoted $df(w)$. The next step is to apply token blocking (②) [22] to form canopies of mentions. Specifically, for each word $w$, a canopy labeled $w$, denoted by $C_w$, is created. Consider an assertion occurrence $a = \langle s; r; o; u \rangle$ in a mention $m$. Let $t(s)$ be the set of non-stop-words contained in the subject name $s$. The mention $m$ is assigned to a canopy $C_w$ if $w \in t(s)$. For example, the subject name of the mention ⟪Charles Dickens; http://. . ./; (wrote; A Tale of Two Cities)⟫ is "Charles Dickens". The mention is thus assigned to two canopies: canopy *Charles* and canopy *Dickens*. After that, clustering (③) is performed on each canopy. GHMS employs HAC in this step: We start with each mention forming a cluster by itself. In each iteration of HAC, the pair of most similar clusters are merged as long as their similarity exceeds a predefined similarity threshold. To perform HAC, we need to define object (i.e., mention) similarity as well as cluster similarity. In [18], a number of mention-similarity functions are evaluated. It is shown that the *IDF token overlap function* results in the most accurate canonicalization. Specifically, given two mentions $m_1$, $m_2$ with subject names $s_1$ and $s_2$, respectively, the similarity function is defined as:

$$sim(m_1, m_2) = \frac{\sum_{w \in t(s_1) \cap t(s_2)} (\log(1 + df(w)))^{-1}}{\sum_{w \in t(s_1) \cup t(s_2)} (\log(1 + df(w)))^{-1}}. \quad (1)$$

Cluster similarity is defined based on single-link clustering, which is shown in [18] to achieve the best canonicalization accuracy compared with other variants, such as complete-link and average-link. Since a mention $m$ is assigned to multiple canopies in Step ② if its subject names contain multiple words, $m$ could involve in multiple clusters. The next step is to merge (④) clusters that share a common mention. For example, in Figure 1(d), mention $m_s$ belongs to clusters 2 and 4. The two clusters are thus merged into cluster 5, as illustrated in Figure 1(e). Finally, the subject names of mentions that are grouped into the same cluster are given a single canonical form (⑤).

## 4 FAC

GHMS is designed with a focus on accurate canonicalization. In particular, experiments are done to evaluate the various HAC variants and various mention-similarity functions in terms of their impacts on canonicalization accuracy. In this paper we focus on how GHMS can be made efficient so that we can canonicalize practically-large OKBs. In this section we discuss various optimization strategies and present our algorithm FAC (**F**ast **A**ssertion **C**anonicalization), which employs those strategies.

We executed GHMS on a number of datasets. The executions were quite slow. For example, it took 41 minutes to canonicalize the small 36K set we mentioned in the introduction. Recall that the 36K set consists of about 36 thousand assertions and is used in [18] to study the canonicalization problem. We also ran GHMS on a

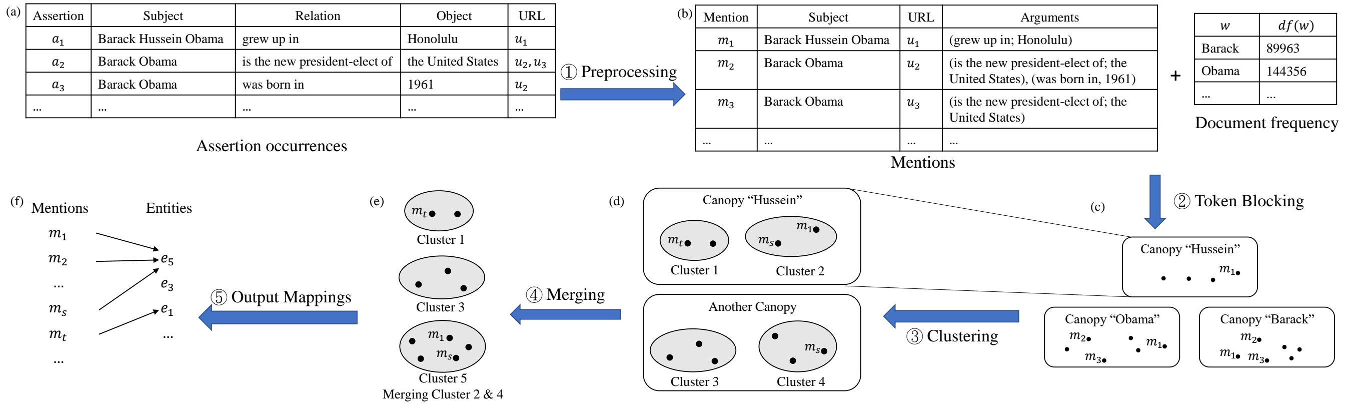


**Figure 1: The GHMS procedure**

dataset with 1.3 million assertions, which took 34 hours. For other larger datasets, GHMS did not finish after 10 days of execution. In particular, GHMS finished clustering a number of canopies that account for only 36% of assertions in a 2.6 million assertions set with 10 days of computation. The reason why GHMS does not scale well to large datasets is that for large sets, canopies are bigger. Clustering mentions in a canopy using HAC is at least quadratic w.r.t. canopy size. In fact, HAC (see Figure 1, step ③) dominates the execution time of GHMS. We conducted profiling to analyze GHMS. We found that even for the very small 36K set, HAC accounted for 99.78% of the total execution time. (Preprocessing ① is a distant second at 0.20%, followed by token blocking ② at 0.008%). Our approach to speed up GHMS thus focuses on making clustering faster.

## 4.1 Efficient Clustering

To cluster mentions in a canopy using HAC, each mention first forms a cluster by itself. Then, iteratively, the two most similar clusters are merged. The process is repeated until the similarity of the two to-be-merged clusters falls below a similarity threshold $\rho$. As mentioned, cluster similarity is based on the single-link definition (which is shown in [18] to be the most effective HAC variant for canonicalization). Our first step is to transform single-link clustering to the problem of finding connected components in a graph.

Definition 2 ($\rho$-Graph). *Given a similarity threshold $\rho$ and a canopy $C$ of mentions, the $\rho$-Graph of $C$ is an undirected graph $G_\rho(C) = (C, E_\rho)$ where the edge set $E_\rho = \{(m_i, m_j) | m_i, m_j \in C \wedge sim(m_i, m_j) \geq \rho\}$. The weight of an edge $(m_i, m_j)$ equals $sim(m_i, m_j)$.*

Given two mentions $m_i$ and $m_j$, if their similarity $sim(m_i, m_j) \geq \rho$, we call the edge $(m_i, m_j)$ a *strong edge*; otherwise, we call it a *weak edge*. The $\rho$-graph of a canopy is therefore composed of all mentions in the canopy as vertices and all and only strong edges between mentions. It can be shown that single-link clustering with a similarity threshold $\rho$ is equivalent to finding connected components in the corresponding $\rho$-graph [21]. In particular, each connected component corresponds to a cluster.

The implementation of GHMS given in [18] follows the iterative merging process. Our optimization techniques, however, apply to the graph-based solution. To distinguish the two realizations of single-link clustering, we use GHMS-g to denote our graph-based connected-component implementation of GHMS.

Given a canopy of $n$ mentions, clustering it with GHMS-g takes two steps: (1) construct the $\rho$-graph and (2) run a breadth-first-search (BFS) algorithm on the $\rho$-graph. Both take $O(n^2)$ time. In fact, the similarities of all $\binom{n}{2}$ mention pairs have to be computed to determine the strong edges in order to construct the $\rho$-graph. In the rest of this section, we present four optimization techniques, namely, *on-demand edge computation*, *inverted index*, *bound by set cardinality*, and *subject name signature*. The first two are *pruning techniques* that prune away unnecessary edges and avoid computing them in the construction of the $\rho$-graph. The last two are *bounding techniques* that provide a very fast estimate of whether an edge is weak without computing the corresponding similarity.

*4.1.1 On Demand Edge Computation (OD).* The general idea of our first technique is to perform BFS to find connected components of a canopy without first constructing the $\rho$-graph in full. Essentially, edges of the $\rho$-graph are discovered incrementally and their weights (i.e., similarities) are evaluated *on-demand*. We use OD to denote this technique. Let us outline how BFS is conducted if edge weights are computed under OD.

First, some notations. Given a vertex (a mention) $v$, let $\mathcal{N}^+(v) = \{u | sim(u, v) \geq \rho\}$ be the set of vertices whose similarities to $v$ equal or exceed the threshold $\rho$. We call $\mathcal{N}^+(v)$ the *strong neighbors* of $v$. Similarly, we use $\mathcal{N}^-(v) = \{u | sim(u, v) < \rho\}$ to denote the set of *weak neighbors* of $v$. We use $cc(v)$ to denote the connected component that contains $v$. Given a canopy $C$ of mentions, BFS under OD starts with a mention (as a vertex) $v_o$ and *explore* a hidden graph to find $cc(v_o)$. During the process, vertices are *explored* one at a time, starting with $v_o$. Let $cc_* \subseteq cc(v_o)$ be a set of $cc(v_o)$'s members that BFS has collected so far. Initially, $cc_* = \{v_o\}$. To *explore* a vertex $v \in cc_*$, we take 2 steps: (1) Determine $\mathcal{N}^+(v) - cc_*$ and add those vertices to $cc_*$. These vertices are the newly found members of the connected component. (2) Add the new members to a queue $Q$. Vertices in $Q$ are explored in subsequent iterations. After the 2 steps, another vertex $v$ is removed from $Q$ and the

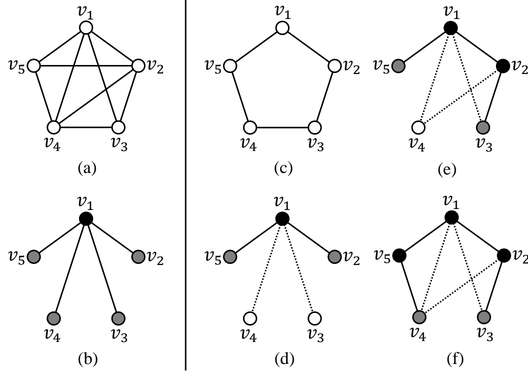


**Figure 2: BFS under OD on a dense $\rho$-graph (a) vs. on a sparse $\rho$-graph (c).**

steps are repeated with the new $v$. When $Q$ becomes empty, $cc_*$ is the desired $cc(v_o)$. If not all vertices in the canopy $C$ have been explored, an unexplored vertex $v'$ from $C$ is picked. The above process is repeated to find another connected component $cc(v')$, and so on.

Note that certain similarities are computed in determining the set $\mathcal{N}^+(v) - cc_*$. Since the graph is hidden (i.e., strong edges are not pre-found), we need to compute the similarities of $v$ to all other vertices in the canopy that are not already collected in $cc_*$ to determine if they are strong or weak. We observe that the number of similarity computations is *fewer* when the connected component is *denser*. To see this, let us consider a simple scenario in which there is only one connected component in the canopy. Figure 2(a) shows a $\rho$-graph with five vertices. The graph is dense (9 out of 10 possible edges are strong). Suppose we start exploring the graph from $v_1$, i.e., $cc_* = \{v_1\}$ initially. We compute the similarities of $v_1$ to all other 4 vertices and find that $\mathcal{N}^+(v_1) = \{v_2, v_3, v_4, v_5\}$. This is illustrated in Figure 2(b), where a line represents a similarity computation. We use a solid (dotted) line to represent a strong (weak) edge computed. In Figure 2(b), all 4 edges computed are strong. Hence, $\{v_2, v_3, v_4, v_5\}$ are all added to $cc_*$. We note that no more similarity computations are needed because $cc_*$ is already the whole canopy. In this example, 4 similarities are computed. Now consider a sparse graph illustrated in Figure 2(c), where the $\rho$-graph forms a ring. Again, we start exploring at $v_1$, 4 similarities are computed as illustrated in Figure 2(d). Two of the computed edges are strong while two are weak. At this point, $cc_* = \{v_1, v_2, v_5\}$ and $v_2$, $v_5$ are queued. Suppose $v_2$ is explored next. Its similarities to $v_3$, $v_4$ are computed because $v_3$ and $v_4$ are not in $cc_*$. These two computations are shown in Figure 2(e). Finally, BFS explores vertex $v_5$, which requires computing the similarity between $v_5$ and $v_4$ (Figure 2(f)). In total, for the ring graph, 7 similarities are computed versus only 4 for the dense graph shown in Figure 2(a). Further note that if the $\rho$-graph is materialized first before BFS is executed, all $\binom{5}{2} = 10$ similarities are computed, regardless of the density of the graph.

In general, consider a canopy that consists of only one connected component of $n$ mentions. If the $\rho$-graph is a complete graph (i.e., very dense), the number of similarity computations is $n - 1$. The fraction of similarity computations under OD (as compared with materializing the $\rho$-graph in full) is $(n-1)/\binom{n}{2} = 2/n$, a very small fraction. On the other hand, if the $\rho$-graph is a ring (i.e., very sparse), we can show that the number of similarity computations is $\binom{n}{2} - (n-2)$. The fraction of similarities computed is thus $[\binom{n}{2} - (n-2)]/\binom{n}{2} \approx 1 - 2/n$, a very large fraction. This illustrates that OD is very effective in pruning similarity computations for dense clusters.

We analyzed different assertion datasets and found that clusters are indeed very dense. For example, we inspected the 10 largest canopies derived from a 2.6M-assertion set. For each such canopy, we inspected its largest cluster. These 10 largest clusters are the most time-consuming clusters to obtain in the canonicalization process. The densities of these clusters range from 46.1% to 99.1% with an average of 76.7%.

In the above analysis, we assume that there is only one connected component (cluster) in a canopy. In practice, a canopy can have multiple connected components. In such a case, each connected component could be just a fraction of the whole canopy. Recall that when we are exploring a vertex $v$, we need to find $\mathcal{N}^+(v) - cc_*$. Under OD, all similarities that involve $v$ except to those vertices in $cc_*$ have to be computed. When a connected component is small compared with the whole canopy, $cc_*$ will be relatively small. This reduces the savings in similarity computations obtained via OD.

To get an idea of the effect of having multiple clusters in a canopy on OD, let us do a back-of-the-envelop calculation. Consider a canopy of $n$ vertices (mentions) that consists of $k$ clusters, each being a complete graph with $n/k$ vertices. Let $F(n, k)$ be the number of similarities OD computes during BFS exploration. From our previous discussion, we know that $F(n, 1) = n - 1$. For $k > 1$, let $v_o$ be the first vertex BFS explores. All similarities from $v_o$ to all other vertices in the canopy are computed, which totaled $n - 1$ computations. Among them, $n/k - 1$ (those from $v_o$ to vertices in $cc(v_o)$) are strong, others (from $v_o$ to vertices in other clusters) are weak. These $n/k - 1$ strong neighbors of $v_o$ will be added to $cc_*$ and to the queue $Q$. When each such strong neighbor $u$ is subsequently removed from $Q$ and explored, the similarities from $u$ to all vertices of other clusters are computed. There are $n - n/k$ such vertices. Since these vertices are from other clusters, their similarities to $u$ are all weak and so they will not be added to $Q$. When $Q$ becomes empty, $cc(v_o)$ is found. Processing all the strong neighbors requires $(n/k - 1) \times (n - n/k)$ similarity computations. Now, there are $n - n/k$ unexplored vertices in the graph distributed across $k - 1$ clusters. The number of similarity computations to complete the BFS is $F(n - n/k, k - 1)$. We thus have the following recurrence,

$$F(n,k) = \begin{cases} n-1, & k = 1 \\ (n-1) + (n/k - 1) \cdot (n - n/k) + F(n - n/k, k-1) & k > 1 \end{cases} \tag{2}$$

When $n \gg k$, Equation 2 is simplified to

$$F(n,k) \approx \begin{cases} n & k = 1 \\ n + (k-1)(n/k)^2 + F(n - n/k, k-1) & k > 1 \end{cases} \tag{3}$$

| Canopy | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| # of Clusters | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 2 | 1 |
| Canopy | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| # of Clusters | 12 | 1 | 5 | 2 | 2 | 1 | 1 | 1 | 4 | 1 |

**Table 1: Number of clusters in each of the top 20 canopies of 2.6M-assertion set (80% coverage)**

Solving the recurrence, we get,

$$F(n,k) = \left(\frac{k+1}{2}\right)n + \left(\frac{k-1}{2k}\right)n^2.$$

The fraction of similarities OD computes compared to that for constructing the $\rho$-graph in full is,

$$\frac{F(n,k)}{\binom{n}{2}} \approx \left[\left(\frac{k+1}{2}\right)n + \left(\frac{k-1}{2k}\right)n^2\right] \Big/ \left(\frac{n^2}{2}\right) = \frac{k+1}{n} + \frac{k-1}{k} \approx \frac{k-1}{k}. \quad (4)$$

For example, if a canopy has two equal-sized clusters (i.e., $k$ = 2), then, we expect OD compute (2-1)/2 = 1/2 of similarities that constructing the complete $\rho$-graph would have done. If there are 3 equal-sized clusters, the fraction goes up to (3-1)/3 = 2/3. Although in practice clusters of a canopy vary in size, and so the assumption we made for the back-of-the-envelop calculation is not valid, the discussion does illustrate that fewer clusters in a canopy implies more OD similarity pruning. We inspected the 20 biggest canopies derived from the 2.6M-assertion set. Table 1 shows the number of clusters that cover 80% of the mentions in each such canopy. We see that except for canopy 11, most mentions of a canopy are concentrated in just one or two clusters. OD should therefore be an effective pruning strategy.

*4.1.2 Inverted Index (INV).* Given two mentions $m_1 = \langle\langle s_1; u_1; Arg_1 \rangle\rangle$ and $m_2 = \langle\langle s_2; u_2; Arg_2 \rangle\rangle$, their similarity $sim(m_1, m_2)$ is given by the IDF token overlap measure (see Equation 1). Recall that given a subject name $s$, $t(s)$ is the set of non-stop-words in $s$. From Equation 1, we see that if $t(s_1) = t(s_2)$, then $t(s_1) \cap t(s_2) = t(s_1) \cup t(s_2) = t(s_1)$, and hence $sim(m_1, m_2) = 1$, the strongest similarity. In this case, the similarity given in Equation 1 needs not be explicitly computed.

Our next optimization strategy is to build an inverted index, $INV$. The index maps a set of words $S$ to a set of mentions $M$ such that the subject name (excluding stop words) of each mention in $M$ is $S$. Formally, $INV(S) = \{m = \langle\langle s; u; Arg \rangle\rangle | t(s) = S\}$. We call $INV(S)$ the *inverted list* of $S$. One can easily verify the following properties of $INV(S)$.

Property 1. $\forall m_i, m_j \in INV(S), sim(m_i, m_j) = 1$.

Property 2. *$\forall m_i, m_j \in INV(S)$, $sim(m_i, m) = sim(m_j, m)$ for any mention m. Hence, m is a strong neighbor of $m_i$ iff m is a strong neighbor of $m_j$.*

Proof.

$$\begin{aligned} sim(m_i, m) &= \frac{\sum_{w \in t(s_i) \cap t(s)} (\log(1 + df(w)))^{-1}}{\sum_{w \in t(s_i) \cup t(s)} (\log(1 + df(w)))^{-1}} \\ &= \frac{\sum_{w \in t(s_j) \cap t(s)} (\log(1 + df(w)))^{-1}}{\sum_{w \in t(s_j) \cup t(s)} (\log(1 + df(w)))^{-1}} \quad (\because t(s_i) = t(s_j)) \\ &= sim(m_j, m) \end{aligned}$$

□

Property 1 implies that mentions in an inverted list form a clique in a $\rho$-graph. Property 2 states that the weight of the edge from a mention $m$ to any mention in the inverted list is the same. Moreover, recall that a canopy $C_w$ is derived for each non-stop-word $w$, and that a mention $m = \langle\langle s; u; Arg \rangle\rangle$ is assigned to canopy $C_w$ if $w \in t(s)$. Since mentions in an inverted list share the same set of non-stop-words, we have:

Property 3. *If a mention $m = \langle\langle s; u; Arg \rangle\rangle$ is in a canopy $C_w$, then all mentions in $INV(t(s))$ are also in $C_w$.*

We modify the preprocessing step (Figure 1, step ①) to include the construction of the inverted index. The index can be used to facilitate clustering a canopy. Specifically, when a vertex $v$ that represents a mention $m = \langle\langle s; u; Arg \rangle\rangle$ in a canopy $C$ is to be explored during the BFS process, we first consult the index and retrieve the inverted list $INV(t(s))$. By Property 3, all mentions in the list must be in the same canopy $C$. Moreover, by Property 1, these mentions are all strong neighbors of $v$. We therefore immediately add the mentions in $INV(t(s))$ to $cc_*$ without having to compute their similarities to $v$. Under OD, these strong neighbors, say, $v'$, should be added to the queue $Q$ (so that they will be explored later). However, we argue that these neighbors, which come from $INV(t(s))$, need not be explored. The reason is that by Property 2, any strong neighbors of, say, $v'$, must be strong neighbors of $v$ too. Hence, all strong neighbors of $v'$ should have already been obtained and included in the connected component when we explored $v$. Therefore, no extra expansion of the connected componet is possible by exploring $v'$. As a result, we do not add mentions in $INV(t(s))$ to the queue for future explorations. This saves much processing costs.

*4.1.3 Bound by Set Cardinality (CARD).* While OD and INV aim at doing *fewer* similarity computations, our next technique, denoted CARD, attempts to make some similarity computations *faster*. From Equation 1, we see that to compute $sim(m_1, m_2)$, we need to look up $df(w)$ for each $w \in t(s_1) \cup t(s_2)$. We can avoid these lookups by computing a simple bound. Let $df_{max}$ and $df_{min}$ be the largest and the smallest $df(w)$ over all non-stop-words $w$ in the dataset. From Equation 1, we have,

$$\begin{aligned} sim(m_1, m_2) &= \frac{\sum_{w \in t(s_1) \cap t(s_2)} (\log(1 + df(w)))^{-1}}{\sum_{w \in t(s_1) \cup t(s_2)} (\log(1 + df(w)))^{-1}} \\ &\le \frac{\sum_{w \in t(s_1) \cap t(s_2)} (\log(1 + df_{min}))^{-1}}{\sum_{w \in t(s_1) \cup t(s_2)} (\log(1 + df_{max}))^{-1}} \quad (5) \\ &= J(t(s_1), t(s_2)) \times (\Gamma)^{-1}, \end{aligned}$$

where $J(t(s_1), t(s_2)) = \frac{|t(s_1) \cap t(s_2)|}{|t(s_1) \cup t(s_2)|}$ is the Jaccard coefficient of $t(s_1)$ and $t(s_2)$; and $\Gamma = \log(1+df_{min})/\log(1+df_{max})$ is a constant. So,

$$(J(t(s_1), t(s_2)) < \rho \cdot \Gamma) \Rightarrow (sim(m_1, m_2) < \rho)\,. \qquad (6)$$

In other words, we only need to compute the Jaccard coefficient and compare it against the constant $\rho \cdot \Gamma$. If the coefficient is smaller than $\rho \cdot \Gamma$, we deduce that mentions $m_1$ and $m_2$ have a weak similarity. In this case, the similarity function is not computed. If the coefficient is not smaller than $\rho \cdot \Gamma$, we compute $sim(m_1, m_2)$ in full.

*4.1.4 Subject Name Signature (SIG).* With CARD, the Jaccard coefficient is computed to test if a similarity is weak. This involves some string matching and subset testing. Our next technique, called SIG, uses signature files to compute a fast upper bound of the Jaccard coefficient to speed up the test. We preprocess each subject name $s$ to obtain an $r$-bit signature $sig(s)$. Specifically, we use a hash function $h(w)$ that maps a word into an integer in the range [0 .. $r$-1]. Let us represent the word $w$ as a sequence of $m$ characters $[c_{m-1}, c_{m-2}, .., c_0]$. The hash function is given as follows:

$$h(w) = \sum_{k=0}^{m-1} 127^k ASC(c_k) \qquad (7)$$

The signature is set as follows:

$$\forall 0 \le i \le r-1, \quad sig(s)[i] = \begin{cases} 1, & \exists w \in t(s) \text{ s.t. } h(w) = i. \\ 0, & \text{otherwise.} \end{cases}$$

, where $ASC(c)$ is the ASCII of character $c$. In our experiments, we set the length of a signature ($r$) to 32.

Given two signatures $sig(s_1)$ and $sig(s_2)$, let $sig_\wedge(s_1, s_2)$ and $sig_\vee(s_1, s_2)$ be the bitwise-AND and bitwise-OR of the two signatures, respectively. We define the Jaccard coefficient of two signatures as

$$J_{sig}(s_1, s_2) = \frac{|sig_\wedge(s_1, s_2)|}{|sig_\vee(s_1, s_2)|}, \qquad (8)$$

where $|\cdot|$ denotes the number of '1' bits in a signature. One can prove the following properties.

Property 4. $|sig_\vee(s_1, s_2)| \le |t(s_1) \cup t(s_2)|$.

Proof. To prove property 4, we need to show that for a word in $t(s_1) \cup t(s_2)$, it would only cause one bit in $sig_\vee(s_1, s_2)$ to be 1. Since the hash function maps each word to one value only, a word can only set a fixed position in a signature to be 1. Thus, $|t(s_1) \cup t(s_2)|$ will be no less than $|sig_\vee(s_1, s_2)|$. If collision occurs, multiple words in $t(s_1) \cup t(s_2)$ are hashed to the same bit in $sig_\vee(s_1, s_2)$. In this case, $|sig_\vee(s_1, s_2)| < |t(s_1) \cup t(s_2)|$. □

Property 5. *If words in $t(s_1) \cap t(s_2)$ do not collide, (i.e., $(w_i \neq w_j) \Rightarrow (h(w_i) \neq h(w_j))\ \forall w_i, w_j \in t(s_1) \cap t(s_2)$), then* $|sig_\wedge(s_1, s_2)| \ge |t(s_1) \cap t(s_2)|$.

Proof. To prove property 5, we show that if $sig_\wedge(s_1, s_2)[i] = x$, then at most one word in $t(s_1) \cap t(s_2)$ is hashed to bit $x$. From the assumption, there is no collision of words in $t(s_1) \cap t(s_2)$. Therefore, if bit $x$ is 1 and there exists a word $w_i$ in $t(s_1) \cap t(s_2)$ s.t. $h(w_i) = x$, then $h(w_j) \neq 1 \forall w_j \in t(s_1) \cap t(s_2) \setminus \{w_i\}$. Note that if bit $x$ is 1 and none of the words $w_i$ in $t(s_1) \cap t(s_2)$ is hashed to $x$, that implies the collision occurs in the words that are not common to both subjects. In this case, $|sig_\wedge(s_1, s_2)| > |t(s_1) \cap t(s_2)|$. □

From the properties, we have,

$$J_{sig}(s_1, s_2) = \frac{|sig_\wedge(s_1, s_2)|}{|sig_\vee(s_1, s_2)|} \ge \frac{|t(s_1) \cap t(s_2)|}{|t(s_1) \cup t(s_2)|} = J(t(s_1), t(s_2)),$$

if words in $t(s_1) \cap t(s_2)$ do not collide. So, $J_{sig}(s_1, s_2)$ is an upper bound of $J(t(s_1), t(s_2))$. If $J_{sig}(s_1, s_2) < \rho \cdot \Gamma$, so is $J(t(s_1), t(s_2))$, and hence $sim(s_1, s_2)$ is weak. Under SIG, we use $J_{sig}(s_1, s_2)$ in place of $J(t(s_1), t(s_2))$ in the test (Equation 6) to identify weak similarities. Since $J_{sig}(s_1, s_2)$ can be computed using bit-wise operations, computing it is much faster than computing $J(t(s_1), t(s_2))$.

Note that using $J_{sig}(s_1, s_2)$ in the test is correct if words in the intersection $t(s_1) \cap t(s_2)$ do not collide. In practice, subject names are not very long and $t(s_1) \cap t(s_2)$ is usually very small. In our datasets, most of these intersections contain 3 or fewer words. Collisions, therefore, rarely occur. If collisions occur, a strong edge may be labeled weak, which could disconnect the connected component and decrease the recall. In our experiments, we obtain the same canonicalization results on a 2.6M assertion set with or without using SIG. Algorithm 1 summarizes the clustering procedure employed by FAC. Line 1 initializes an array to record if a vertex is visted. Line 2 initialize a queue to support BFS and an integer $i$ to label the connected components. Line 3-4 iterates all vertices to ensure that every vertex is labeled with a connnected component. Line 5-15 is the BFS algorithm. In line 8-9, we use INV to find all mentions that have exact subject names as $m_2$, the mention which is being explored. These mentions need not be explored according to Property 2 and therefore are removed from Q. In line 12, SIG is applied to compute the upper bound of the similarity between $m_2$ and its neighbor $m_4$. If the upper bound of a mention pair is less than the threshold, we ignore such mention pair. Otherwise, the exact similarity score (line 13) needs to be computed to determine whether $m_4$ is similar to $m_2$.

# 5 EXPERIMENTS

In this section we present experimental results evaluating the canonicalization algorithms' performances. First, a quick review of the algorithms: GHMS performs canopy-based single-link HAC, which is achieved using an iterative cluster-merging approach. In our experiment, we use the implementation of GHMS kindly provided by the authors of [18]. GHMS-g is our modification of GHMS, in which HAC is done by finding connected components in a $\rho$-graph (see Section 4.1). Finally, FAC is based on GHMS-g with the four optimization techniques OD, INV, CARD and SIG employed. All algorithms are implemented in Java 7. Experiments are conducted on a machine with an Intel i7-6850K CPU and 128GB RAM running Ubuntu 16.04.3. We follow [18] and set $\rho = 0.5$.

## 5.1 Datasets

We evaluated the algorithms on 15 assertion datasets, all of which are extracted from the ClueWeb09 corpus using ReVerb. Table 2 lists these datasets and their corresponding statistics. Here, we briefly describe the datasets. The **15M set** is a dataset provided by the ReVerb project [14]. It consists of 15 million assertions. In [18], several filtering schemes are applied to the 15M set. (For example,

**Algorithm 1:** Cluster a canopy

**Input:** A set of mentions $M$; Signatures $sig(\cdot)$'s; Inverted index $INV(\cdot)$; Similarity threshold $\rho$; Constant $\Gamma$.
**Output:** Cluster id for each mention in the canopy.

```
Explored[ ] = {0, 0, ..., 0} ;  // Explored[i] = mention i's cluster id
Queue Q; i = 0;
for Every mention m1 in M do
    if Explored[m1] ≠ 0 then
        Q.enqueue(m1); i++;
        while Q is not empty do
            m2 = Q.dequeue; Explored[m2] = i;
            for Every mention m3 in INV(m2) do
                Explored[m3] = i; Q.remove(m3);
            for Every mention m4 in M do
                if Explored[m4] ≠ 0 then
                    if J_sig(m2.subject, m4.subject) ≥ ρ × Γ then
                        if sim(m2, m4) ≥ ρ then
                            Q.enqueue(m4); Explored[m4] = i;
return Explored;
```

assertions whose subjects do not contain any proper nouns are removed.) This filtered set contains 2.6 million assertions. We call this set the **2.6M set**. In [18], the 2.6M set is further filtered and sampled using information from Freebase to obtain 36 thousand assertions (see [18] for details). We call this set the **36K set**[2]. Both the 2.6M and the 36K sets are provided by [18].

Based on the 2.6M set, we further create 12 sets, which are divided into two groups of 6 sets each. The first group, called the *F-group*, are denoted by *FxK*, where "$x$" ranges from 100 to 1,300. The group is obtained as follows: We sort the assertions in the 2.6M set in alphabetical order of their subject names. The first $x$ thousand assertions in the sorted list are collected into the set *FxK*. The second group, called the *S-group*, are denoted by *SxK*. To create the *S-group*, we first randomly select 1,300 thousand assertions from the 2.6M set to form the *S1300K* set. This set is then downsampled to 500K assertions, which is then further downsampled to 400K assertions, and so on. Note that a smaller *F-group* set is a proper subset of a larger *F-group* set; likewise for the *S-group*.

Each row in Table 2 shows the statistics of a dataset. In particular, it shows the size of the largest canopy expressed in number of mentions (column e); the average density of a cluster, measured by the number of strong edges in a cluster as a fraction of the total number of possible edges of the cluster (column g); and the number of canonicalized forms (i.e., entities) identified (column i).

We remark that an *F-group* set is obtained from the subject-name-sorted 2.6M set. Assertions in an *F-group* set are therefore *more focused* on a smaller number of entities compared with those of an *S-group* set. This is reflected by the fewer and larger canopies for *F-group* sets compared with *S-group* sets (see columns e and f).

[2] The set is called *Ambiguous* in [18].

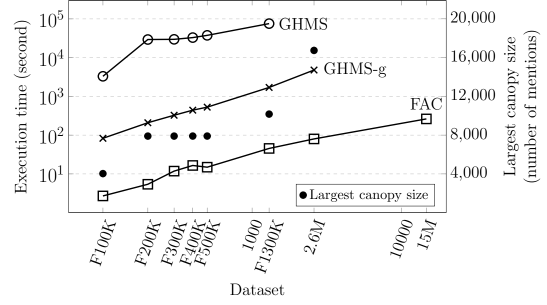


**Figure 3: Execution times (*F-group*, 2.6M, and 15M sets)**

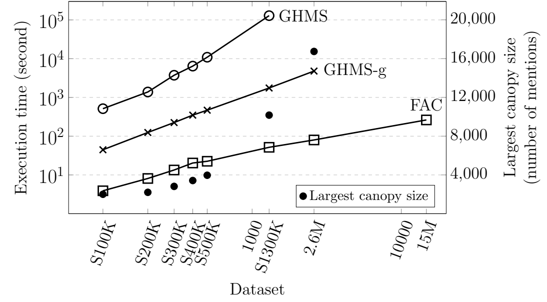


**Figure 4: Execution times (*S-group*, 2.6M, and 15M sets)**

## 5.2 Results

We start with running the algorithms on the small 36K set, which is used in [18] to illustrate canonicalization. It takes GHMS 2,477 seconds (~41 minutes) to complete, while FAC takes only 0.2 seconds. We will focus on the larger datasets for the rest of this section.

*5.2.1 Algorithms Comparison.* Figure 3 shows the execution times of GHMS, GHMS-g and FAC when they are applied to canonicalize the *F-group*, 2.6M, and 15M sets. The datasets are labeled on the $x$-axis according to their sizes. The labels "1000" and "10000" refer to sizes of 1000K and 10000K assertions, respectively. They are shown as size references on the $x$-axis. Note that both $x$ and $y$ axes are in log scale. Figure 4 is a similar figure showing the algorithms' performance on the *S-group* sets. From the figures, we make a few observations:

• **FAC is orders-of-magnitude faster than GHMS and GHMS-g.** In particular, for the 2.6M and 15M sets, GHMS does not terminate after 10 days of execution. The long execution times is due to time-consuming HAC, which is applied to cluster each canopy. On closer inspection, we found that, for GHMS, this clustering time is cubic w.r.t. the number of mentions in a canopy; For GHMS-g, which reduces HAC to finding connected components in a $\rho$-graph, the clustering time is quadratic because the similarities of all possible edges in the $\rho$-graph have to be computed to construct the graph. Tables 3(a) and (b) show the speedups of FAC against GHMS and GHMS-g, respectively. We see that FAC registers up to 4-orders-of-magnitude speedups against GHMS and up to 2-orders-of-magnitude speedups against GHMS-g. FAC is very efficient. In particular, for our largest 15M set, FAC takes only about 4.4 minutes to complete.

| | Dataset | (a) # assertions | (b) occurrences | (c) # mentions | (d) # canopies | (e) largest canopy size | (f) avg # cluster per canopy | (g) avg density of clusters | (h) avg length of inverted list | (i) # of canonicalized forms (entities) |
|---|---|---|---|---|---|---|---|---|---|---|
| | 15M | 14,728,268 | 25,414,939 | 24,429,156 | 393,972 | 134,596 | 3.06 | 56.1% | 18.18 | 520,060 |
| | 2.6M | 2,602,621 | 4,431,106 | 4,291,251 | 131,598 | 16,738 | 4.51 | 62.4% | 9.41 | 190,713 |
| | 36K | 35,630 | 36,853 | 34,209 | 460 | 3,469 | 1.36 | 98.9% | 64.40 | 349 |
| F-Group | F100K | 100,000 | 191,047 | 181,161 | 17,279 | 4,030 | 2.99 | 81.4% | 6.21 | 18,934 |
| | F200K | 200,000 | 365,668 | 349,800 | 26,430 | 7,900 | 3.46 | 83.3% | 6.77 | 32,675 |
| | F300K | 300,000 | 534,475 | 510,687 | 33,832 | 7,900 | 3.84 | 77.5% | 6.99 | 45,681 |
| | F400K | 400,000 | 702,536 | 672,621 | 40,426 | 7,903 | 4.17 | 72.4% | 7.02 | 58,198 |
| | F500K | 500,000 | 862,668 | 825,541 | 46,500 | 7,903 | 4.24 | 70.2% | 7.14 | 69,917 |
| | F1300K | 1,300,000 | 2,193,970 | 2,104,238 | 85,111 | 10,169 | 4.86 | 54.7% | 7.72 | 141,049 |
| S-Group | S100K | 100,000 | 173,450 | 171,329 | 32,739 | 2,003 | 4.33 | 67.6% | 2.74 | 41,934 |
| | S200K | 200,000 | 344,685 | 339,914 | 47,383 | 2,204 | 4.69 | 60.7% | 3.19 | 63,819 |
| | S300K | 300,000 | 512,838 | 504,753 | 57,882 | 2,816 | 4.82 | 55.4% | 3.54 | 79,916 |
| | S400K | 400,000 | 682,546 | 671,239 | 66,188 | 3,421 | 4.90 | 54.2% | 3.88 | 91,887 |
| | S500K | 500,000 | 850,631 | 835,468 | 73,003 | 3,966 | 4.93 | 54.7% | 4.18 | 101,759 |
| | S1300K | 1,300,000 | 2,213,163 | 2,153,310 | 105,897 | 8,707 | 4.82 | 58.6% | 6.36 | 149,209 |

**Table 2: Dataset Statistics**

| F100K | F200K | F300K | F400K | F500K | F1300K | 2.6M | 15M |
|---|---|---|---|---|---|---|---|
| 1,234 | 5,485 | 2,515 | 1,997 | 2,524 | 1,661 | -- | -- |
| S100K | S200K | S300K | S400K | S500K | S1300K | | |
| 131 | 172 | 279 | 317 | 483 | 2,383 | | |

**(a) FAC speedups against GHMS**

| F100K | F200K | F300K | F400K | F500K | F1300K | 2.6M | 15M |
|---|---|---|---|---|---|---|---|
| 31.0 | 29.0 | 27.7 | 26.8 | 35.4 | 37.4 | 60.0 | -- |
| S100K | S200K | S300K | S400K | S500K | S1300K | | |
| 11.4 | 15.5 | 16.8 | 17.2 | 20.8 | 34.0 | | |

**(b) FAC speedups against GHMS-g**

**Table 3: FAC speedups against (a) GHMS and (b) GHMS-g**

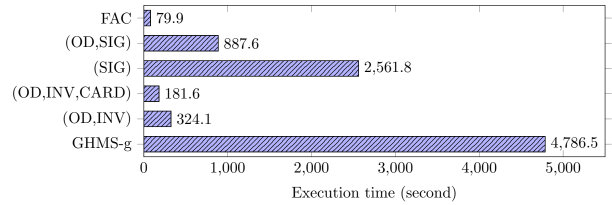


**Figure 5: Ablation analysis (2.6M set)**

• **GHMS shows very different performances over the F-group sets and the S-group sets.** We observe that the performance curves of GHMS are very different in Figures 3 and 4. For example, GHMS's execution time stays high and flat from the *F200K* set to the *F500K* set, while it increases gradually from the *S200K* set to the *S500K* set. To understand this difference, we mark the largest canopy size (i.e., Table 2 column e) for each dataset with • in the figures. (Values are shown on the right $y$-axes.) As we have discussed, GHMS applies cubic HAC to cluster mentions in a canopy. Its total execution time is thus dominated by the clustering time of the largest canopy in a dataset. For the *F-group*, it happens that a large canopy with 7,900 mentions is created in the *F200K* set and no significantly larger canopies are created as we move to *F500K*. GHMS's execution time through these sets therefore stays high and flat. From Figures 3 and 4, we observe a close relation between the largest canopy size and GHMS's execution time, due to the time being dominated by the largest canopy.

• **FAC gives larger speedups for the F-group sets than for the S-group sets.** From Tables 3(a) and (b), we see that FAC's speedups are generally much larger for the *F-group*. For example, FAC's speedups over GHMS are 1,234x for *F100K* and 131x for *S100K*. The reasons for this big difference are threefold. Firstly, as explained in our previous observation, GHMS's execution is much dominated by the biggest canopy. From Table 2, column (e), we see that the biggest canopies in the *F-group* sets are generally much bigger than those of the *S-group* sets. GHMS is therefore much slower executing on the *F-group*. Secondly, OD is most effective when there are few but dense clusters in a canopy (Section 4.1.1) From Table 2, columns (f) and (g), we see that in general, canopies in an *F-group* set contain fewer but denser clusters than those in an *S-group*. Hence, OD is more effective for the *F-group*. Thirdly, From Table 2, column (h), we see that inverted lists in the *F-group* are generally longer than those in the *S-group*. This makes INV more effective (Section 4.1.2) for the *F-group*. These result in much higher speedups when FAC is applied to *F-group* sets.

• **FAC is scalable.** By applying a graph-based solution to perform HAC, we reduce the clustering time from cubic to quadratic. Moreover, by applying all the pruning and bounding strategies, FAC avoids much of the similarity computation. For example, from our discussion in Section 4.1.1, by applying the OD technique alone, if a canopy consists of only one dense cluster, then only a linear number of similarities are computed. In Figures 3 and 4, if we consider FAC's curves as straight lines, the slopes of the lines are 0.92 and 0.83, respectively. Since the figures are log-log plots, this indicates that FAC's execution time is sub-linear (or close to linear) in data size. FAC can thus scale well to very large datasets.

5.2.2 *Ablation Analysis.* Next, we perform an ablation analysis on FAC to evaluate the pruning and bounding techniques. Note that while OD and INV are orthogonal pruning techniques, which

can be applied at the same time, we can choose only either CARD or SIG as the bounding test. We use the 2.6M set in this experiment. Figure 5 shows the execution time of FAC as well as those when some techniques are removed. Specifically, we have

**FAC**: OD, INV as pruning, SIG as bounding;
**(OD, SIG)**: OD as pruning, SIG as bounding, i.e., INV is turned off;
**(SIG)**: SIG as bounding, no pruning is done;
**(OD, INV, CARD)**: OD and INV as pruning, CARD as bounding;
**(OD, INV)**: OD and INV as pruning, no bounding is done;
**GHMS-g**: No pruning or bounding.

From Figure 5, we make the following observations:

- Comparing FAC and (SIG), we see that the execution time increases by a factor of 32 (from 79.9s to 2561.8s). This shows that the pruning techniques OD and INV are highly effective and are essential for efficient canonicalization.
- Comparing FAC and (OD, SIG), the execution time increases by ~11x (from 79.9s to 887.6s). This shows that out of the 32x speedup brought about by pruning, a factor of 11x is contributed by the inverted index INV.
- Comparing FAC and (OD, INV), the execution time increases by ~4x (from 79.9s to 324.1s). This shows that SIG contributes a significant 4x speedup as a bounding technique. If we use CARD for bounding instead (i.e, we compare (OD, INV, CARD) vs. (OD, INV)), the execution time increases by ~ 1.8x only (from 181.6s to 324.1s). This shows the advantage of SIG over CARD, as SIG's bounding test can be evaluated quickly with bitwise operations.
- Comparing (SIG) and GHMS-g, the execution time increases by ~1.8x (from 2561.8s to 4786.5s). This shows that if no pruning is done, then the bounding technique SIG can only achieve a 1.8x speedup. This observation, compared with the previous one (in which SIG achieves a 4x speedup when pruning with OD and INV is applied), is interesting — it shows that there is synergy between our pruning and bounding techniques. For example, given a mention $m$, we use the inverted index to find all other mentions whose similarities to $m$ are 1 (see Section 4.1.2, Property 1). All similarities among these mentions, which are all strong ones, are "vacuously" determined (and thus these similarity computations are pruned). Recall that the objective of bounding is to apply a fast test to find out weak similarities. Because many of the strong similarities are found and filtered by INV, those that are left for the bounding test are more likely to be weak. This makes saving by bounding more effective.

*5.2.3 Pruning and Bounding Effectiveness.* Table 4 (a) shows the pruning effectiveness of OD and INV when they are applied to canonicalize the 2.6M set. There are about 4.57 billion mention similarities that are needed to construct all the $\rho$-graphs of the canopies. (Those are the similarities GHMS-g would need to compute.) With OD pruning, about 2.24B of the similarities are computed. OD's pruning effectiveness is (4.57-2.24)/4.57 = 42.6%. By employing an inverted index, the number of similarities is further reduced to about 0.22B — about 95.3% of the similarities are pruned. Our pruning strategies are therefore highly effective.

Table 4(b) shows the effectiveness of CARD and SIG. Recall that the objective of a bounding method is to quickly evaluate a bounding test to identify weak mention similarity. We thus define a bounding technique's effectiveness as the fraction of weak similarities that satisfy the condition of the bounding test (e.g., Equation 6 for CARD). We consider two cases: with pruning on (Table 4(b), column (a)) and with pruning off (column (b)). From the table, we see that the bounding techniques are highly effective. In particular, with pruning, 98.5% of the weak similarities are "captured" by CARD's bounding test without computing the similarities exactly. Although SIG is slightly less effective than CARD, it leads to a better FAC efficiency (see Section 5.2.2) because the signature-based tests are evaluated more efficiently. Moreover, we observe that pruning improves bounding effectiveness. This again shows the synergy between pruning and bounding. We observe similar pruning and bounding effectiveness for other datasets experimented in our study.

*5.2.4 Case Study.* In this section we briefly mention an example to illustrate an application of canonicalization. We apply FAC on the 2.6M set and retrieve a cluster whose subject's canonical form is "UPS and FedEx". There are 97 assertions in this cluster. We manually inspect all these assertions and find that 50 of them are redundant in the sense that they carry the same information as other assertions in the set. For example,
$\langle$*UPS and FedEx*; *cannot deliver to*; *P.O. Boxes*$\rangle$
is a redundant assertion because it semantically replicates
$\langle$*FedEx and UPS*; *can not ship to*; *Post Office Boxes*$\rangle$.
The redundancy in this cluster is more than 50%. An interesting question is how redundant assertions can be effectively found. We attempt to answer this question by canonicalizing the *object names* of assertions as well. Now, we consider two assertions $a_1 = \langle s_1; r_1; o_1 \rangle$ and $a_2 = \langle s_2; r_2; o_2 \rangle$ to be the same if $f(s_1) = f(s_2)$, $r_1 = r_2$ and $f(o_1) = f(o_2)$ (i.e., their subjects/objects share the same canonical forms and they have the same relation name). By this means, we are able to find 24 of the 50 redundant assertions. Note that the two real assertions shown above are not detected as duplicates by the simple method because they have different relation names. Potentially, we could achieve even more effective duplicate detection if we canonicalize relation name as well. Nonetheless, this case study gives us some interesting insights. (1) Assertions could be highly redundant in an OKB. This unnecessarily increases the storage and processing requirements in knowledge management and processing. (2) Entities in assertions could be composite *virtual* entities. In this case study, "UPS and FedEx" is not a single physical entity; it is more of a concept of "representative delivery services". It may not be easy to link such virtual entities to those in a curated KB. (3) Canonicalization can help us detect redundancy and identify virtual entities. This has the potential of significantly improving the performance of an OKB system in answering queries.

# 6 CONCLUSIONS

In this paper we studied the problem of efficient canonicalization of large OKBs. We put forward the FAC algorithm, which applies various pruning and bounding techniques to avoid mention similarity computations. We provided in-depth analysis on our techniques. In particular, we show that the pruning strategies OD and INV are particularly effective for datasets whose canopies contain few but dense clusters, and whose inverted indices contain long inverted lists. Through extensive experiments, we evaluated FAC over a

| | Total | w/ OD | w/ (OD + INV) |
|---|---|---|---|
| Similarities computed: | 4,572.8M | 2,241.3M | 215.0M |
| Pruning effectiveness: | | 42.6% | 95.3% |

**(a) Pruning effectiveness**

| | | Pruning (OD + INV) | |
|---|---|---|---|
| | | (a) Enabled | (b) Disabled |
| | Total # of weak similarities: | 212M | 3,081M |
| CARD | Weak similarities computed: | 3.2M | 529M |
| | Bounding effectiveness: | 98.5% | 82.5% |
| SIG | Weak similarities computed: | 21.3M | 751M |
| | Bounding effectiveness: | 90.0% | 75.6% |

**(b) Bounding effectiveness**

**Table 4: Pruning and bounding effectiveness (2.6M set)**

number of large datasets. Our results show that our strategies are highly effective and FAC is scalable. FAC is thus a feasible solution to canonicalize practically-large OKBs.

## ACKNOWLEDGMENTS

We thank the authors of [18] for providing us their code and datasets.

## REFERENCES

[1] I. Bhattacharya and L. Getoor. 2006. A latent dirichlet model for unsupervised entity resolution. In *ICDM*. SIAM, 47–58.
[2] L. Del Corro and R. Gemulla. 2013. ClausIE: clause-based open information extraction. In *WWW*. ACM, 355–366.
[3] A. Carlson et al. 2010. Toward an Architecture for Never-Ending Language Learning. In *AAAI*, Vol. 5. 3.
[4] A. McCallum, et al. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*. ACM, 169–178.
[5] B. Hachey, et al. 2013. Evaluating entity linking with Wikipedia. *Artificial Intelligence* 194 (2013), 130–150.
[6] K. Bollacker, et al. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*. ACM, 1247–1250.
[7] L. A. Galárraga, et al. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*. ACM, 413–422.
[8] Mausam et al. 2012. Open Language Learning for Information Extraction. In *EMNLP-CoNLL*. ACL, 523–534. http://www.aclweb.org/anthology/D12-1048
[9] M. Banko, et al. 2007. Open Information Extraction from the Web. In *IJCAI*, Vol. 7. 2670–2676.
[10] M. Dredze, et al. 2010. Entity disambiguation for knowledge base population. In *COLING*. ACL, 277–285.
[11] O. Benjelloun, et al. 2009. Swoosh: a generic approach to entity resolution. *VLDB* 18, 1 (2009), 255–276.
[12] P. Yin et al. 2015. Answering Questions with Complex Semantic Constraints on Open Knowledge Bases. In *CIKM*. ACM, 1301–1310.
[13] S. Auer, et al. 2007. Dbpedia: A nucleus for a web of open data. *The Semantic Web* (2007), 722–735.
[14] A. Fader, S. Soderland, and O. Etzioni. 2011. Identifying relations for open information extraction. In *EMNLP*. ACL, 1535–1545.
[15] A. Fader, L. Zettlemoyer, and O. Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In *KDD*. ACM, 1156–1165.
[16] J. Fisher, P. Christen, Q. Wang, and E. Rahm. 2015. A clustering-based framework to control block sizes for entity resolution. In *KDD*. ACM, 279–288.
[17] E. Gabrilovich, M. Ringgaard, and A. Subramanya. 2013. FACC1: Freebase annotation of ClueWeb corpora. http://lemurproject.org/clueweb09/FACC1/. (2013).
[18] L. A. Galárraga, G. Heitz, K. Murphy, and F. M. Suchanek. 2014. Canonicalizing open knowledge bases. In *CIKM*. ACM, 1679–1688.
[19] J. Krishnamurthy and T. M. Mitchell. 2011. Which noun phrases denote which concepts?. In *HLT*, Vol. 1. ACL, 570–580.
[20] T. Lin and O. Etzioni. 2012. Entity linking at web scale. In *AKBC-WEKEX*. ACL, 84–88.
[21] C. D. Manning, P. Raghavan, and H. Schütze. 2008. *Hierarchical Clustering*. Cambridge University Press, 346–368. https://doi.org/10.1017/CBO9780511809071.018
[22] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. 2011. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*. ACM, 535–544.
[23] L. Ratinov, D. Roth, D. Downey, and M. Anderson. 2011. Local and global algorithms for disambiguation to wikipedia. In *HLT*, Vol. 1. ACL, 1375–1384.
[24] W. Shen, J. Wang, and J. Han. 2015. Entity linking with a knowledge base: Issues, techniques, and solutions. *TKDE* 27, 2 (2015), 443–460.
[25] F. M. Suchanek, G. Kasneci, and G. Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*. ACM, 697–706.
[26] A. P. Yates and O. Etzioni. 2009. Unsupervised methods for determining object and relation synonyms on the web. *JAIR* (2009).