Hierarchical Entity Resolution using an Oracle

Sainyam Galhotra
University of Chicago
sainyam@uchicago.edu

Donatella Firmani
Sapienza University
donatella.firmani@uniroma1.it

Barna Saha
UC San Diego
bsaha@eng.ucsd.edu

Divesh Srivastava
AT&T Chief Data Office
divesh@research.att.com

ABSTRACT

In many applications, entity references (i.e., records) and entities need to be organized to capture diverse relationships like type-subtype, is-A (mapping entities to types), and duplicate (mapping records to entities) relationships. However, automatic identification of such relationships is often inaccurate due to noise and heterogeneous representation of records across sources. Similarly, manual maintenance of these relationships is infeasible and does not scale to large datasets. In this work, we circumvent these challenges by considering weak supervision in the form of an oracle to formulate a novel *hierarchical* ER task. In this setting, records are clustered in a tree-like structure containing records at leaf-level and capturing record-entity (duplicate), entity-type (is-A) and subtype-supertype relationships. For effective use of supervision, we leverage triplet comparison oracle queries that take three records as input and output the most similar pair(s). We develop `HierER`, a querying strategy that uses record pair similarities to minimize the number of oracle queries while maximizing the identified hierarchical structure. We show theoretically and empirically that `HierER` is effective under different similarity noise models and demonstrate empirically that `HierER` can scale up to million-size datasets.

CCS CONCEPTS

• **Information systems** → **Entity resolution**; *Entity relationship models.*

KEYWORDS

Ontology construction, type hierarchy, entity resolution

ACM Reference Format:
Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. 2022. Hierarchical Entity Resolution using an Oracle. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3526147

1 INTRODUCTION

In many applications, records are represented in diverse formats like images, unstructured and structured text and these records need to be organized to capture complex relationships. For example, e-commerce websites like Amazon organize product listings from different sellers by identifying duplicate products, which are shown under the 'other sellers' option in Figure 1 and arrange them in the form of a taxonomy to enable better search and recommendations. In this example, the different listings in the other sellers category denote the record-entity relationship (also known as co-reference or duplicate relationship), type categorization of a product at the top of the webpage denotes the is-A relationship and the type hierarchy denotes the type-subtype or hypernym relationship.

The process of de-duplication or entity resolution identifies record-entity relationships and has been widely used for data integration and cleaning [18] for more than 50 years. Hierarchical arrangement of types based on hypernym and is-A relationships is useful to construct taxonomies or ontologies for knowledge graph construction and management. Assigning records to the identified taxonomy is useful for diverse applications like recommendation [39], categorization [47], and search [67].

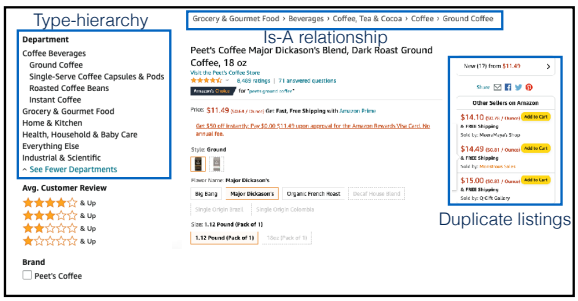


Figure 1: Example product listing with duplicates on the right, is-A relationship mapping record to the type node on the top and subtype-supertype hierarchy on the left.

Record id	Title	Price	Brand
1	Narasu's / Narasus Extra Strong Spray Dried Instant Coffee 100g	11.99	Narasu
2	Hawaii Selection/ Ice Coffee 100% Kona/ Spray Dried Instant/ 1.5 oz (43g)	15.99	
3	Peet's Coffee Major Dickason's Blend, Dark Roast Ground Coffee, 18 oz	9.98	Peet
4	Twinings English Breakfast Tea Bags	12.99	Twinings
5	Taylors of Harrogate English Breakfast, 20 Teabags	34.95	Taylors
6	Native Organic Instant Freeze Dried Coffee, 3.17 Oz (Pack Of 2)	11.49	Native
7	One Love Tea – Berry Bliss Yerba Mate – 3 Oz Loose Leaf Tea Tisane	12.99	One Love
8	Waka Coffee Quality Instant Coffee, Colombian, Freeze Dried	9.26	Waka
9	Peet's Coffee Dickason's Blend, Dark Roast Whole Beans	9.99	Peet
10	Bigelow Caffeinated Green Tea, 40-count	21	Bigelow
11	Twinings Breakfast Tea, 100 Count	5.33	Twinings
12	Twinings of London English Breakfast Tea Bags, 100 Count	24.99	Twinings
13	Bigelow Green Tea with Lemon Tea Bags 28-Count Boxes	24.74	Bigelow
14	Stash Tea Fruity Herbal Tea	9.99	Stash

Figure 2: Example collection of products along with a seed type-hierarchy constructed by a domain expert.

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526147

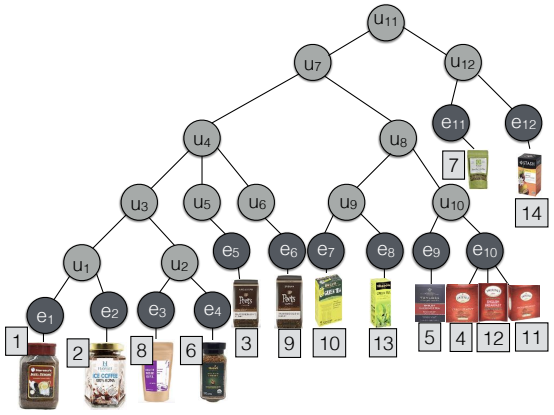


Figure 3: Ground truth hierarchy with dark gray nodes denoting entities and light gray nodes denoting type-nodes. Some internal nodes are: u_{11} = beverages, u_7 = caffeinated beverages, u_{12} = non-caffeinated beverages.

Even though the benefits of constructing a type hierarchy are widely acknowledged, in practice the majority of subtype-supertype relationships are maintained manually by domain experts. Manual maintenance of such relationships is labor-intensive as the type hierarchy evolves whenever new products are introduced/discontinued. Automatic identification of type nodes and is-a relationships in the hierarchy is also challenging due to presence of noise and heterogeneity of representation across data sources (sellers in the Amazon example). We demonstrate some hierarchical ER challenges with the following example.

Example 1.1. Consider a retail website that sells different grocery items provided by different sellers. Figure 2 shows images and textual descriptions of 14 records consisting of different types of beverages. To categorize these products, a domain expert manually constructs a simple type-subtype based hierarchy that contains tea/coffee where the coffee products can be further categorized into whole bean, ground and instant coffee (Figure 2). However, this hierarchy is incomplete and does not contain the following information. (a) Tea products can be further categorized as black tea and green tea. (b) Instant coffee products can be categorized into spray-dried and freeze-dried instant coffee. (c) Some products are duplicates of each other, e.g., records 4, 11 and 12 refer to Twinings English breakfast tea and assigning all these records as siblings of record 5 loses this information. Figure 3 presents the ground truth hierarchy that captures all relationships with product records as leaf nodes, dark gray nodes as entity nodes (all records under this internal node refer to same entity), and light gray nodes as type nodes (capturing hypernym relationships). Among the type nodes, u_3–u_6, correspond to the internal nodes denoting different types of coffee in the initial hierarchy constructed by the expert.

We now focus on this example to discuss the challenges of prior automated and manual techniques to construct such hierarchies.

Automatic Construction. To automatically construct a set of types and subtype-supertype relationships, prior techniques have proposed to leverage co-occurence patterns of hypernyms [37, 52]. However, certain type-nodes are often not mentioned in textual descriptions of the records [48] and if the records are collected from different sources, each source uses a different terminology. For example, the type-node 'non-caffeinated beverages' is never mentioned in any of the product descriptions in Figure 2. Similarly, none of the breakfast teas mention 'black tea' in any of the titles. Moreover, certain tokens like "hazelnut" in "hazelnut coffee" denote attributes like flavor and not a hypernym relationship.

Manual Construction. A domain expert can easily generate a hierarchy that contains well-known categories. However, such a hierarchy may not capture all hypernym relationships. To determine specialized type-nodes like spray-dried and freeze-dried instant coffee in Example 1.1, a domain expert has to be aware of all variations for different product types. Manual processing of millions of records to identify their product types and such variations is infeasible. Moreover, introduction of a new product may require new types node in the hierarchy. For example, new internal nodes like 'caffeinated' and 'non-caffeinated' beverages are added when tisane/herbal teas (records 7 and 14) are added to the dataset and the hierarchy. Manual maintenance and evolution of hypernym relationships is not scalable. However, if only three records are considered in isolation, say 4, 5, and 13, then a user (or a trained classifier) can easily distinguish that 4 and 5 are closer to each other than either of them is to 13. Answering such queries does not require the context of all variations of types in the constructed hierarchy. Even though considering record triplets in isolation can help uncover the hierarchical structure, it is not scalable to compare all possible triples for million scale datasets.

To handle the limitations of a fully automated or a fully manual construction, and the challenges of dealing with new products and evolving hierarchies, we propose to study the problem of organizing records in the form of a hierarchy that enriches the initial seed hierarchy (provided by the domain expert)by identifying duplicate records, and enriching it with is-A and new subtype-supertype relationships. We call this problem as the *Hierarchical ER* problem. To deal with the challenges of automated techniques, we propose an oracle-based approach to compare triplets. An oracle is an abstraction of a classifier (learned using active-learning based techniques in the absence of training data) or a domain expert that answers two types of queries, given below. The recent advancements of leveraging deep learning based classifiers for ER [11, 20, 46, 49] and related tasks are also alternative implementation of the oracle.

- "do records u and v refer to the same entity?" and
- "which pair of records among u, v and w are most similar?"

Such comparisons reveal the local hierarchical structure with respect to the queried records and can be answered without the knowledge of other records in the dataset. These oracle models have been widely popular to study fairness metrics [40], correlation clustering [59] and classification [38, 57], identify maximum elements [34, 61], top-k elements [13, 17, 19, 43, 44, 54], information retrieval [42], skyline computation [62], and so on. In order to minimize the oracle workload, our framework prioritizes records to optimize the number of triplet comparisons.

Related problems and solution techniques. The closest task in the literature to our problem is Entity Resolution (ER). ER has been studied for more than 50 years and constitutes a fundamental component of data integration pipelines [18]. ER typically focuses on identifying which records refer to the same entity, and ignores

type information. Some prior ER techniques leverage type information of entities to improve ER classification but do not study the identification of record types for hierarchical arrangement [73, 74]. Another related task in the literature is Hierarchical Clustering. This has been studied in a variety of application domains including the construction of phylogenetic trees (ontologies of animal or plant species) [10, 21, 41] and taxonomies [56, 70, 72]. In such applications, records refer to different entities, and thus clustering methods ignore entity resolution. Moreover, these techniques build almost binary hierarchies, where every node has two (or slightly more than two) children. Thus, neither ER nor Hierarchical Clustering techniques can by itself solve our problem effectively.

Limitations of using existing strategies. One approach to solve hierarchical ER could be to run ER first followed by hierarchical clustering (or vice versa). However, pipelining the two processes turns out to be sub-optimal. Let n be the number of records.

- Running a hierarchical clustering technique like [21] first and then post-processing the bottom level in order to detect entities can require $O(n^2)$ queries for non-binary hierarchies in the worst case, before even identifying the entities.
- Running a pairwise matching based ER technique like [24] first and post-processing entities after that to detect types can be efficient in case of large entities but can require $O(n^2)$ queries to identify small entity clusters, before even starting to process types.

Even in the case where all records refer to distinct entities, prior oracle-based hierarchical clustering techniques [21] require $O(n^2)$ queries for non-binary hierarchies.

Our contributions. We develop two algorithms for Hierarchical Entity Resolution using an Oracle, called `Hier-Type` and `HierER`,

- `Hier-Type` outperforms hierarchical clustering methods by dealing effectively with type-only hierarchies (i.e., all entities are of size one) with arbitrary degree distribution;
- `HierER` solves the hierarchical ER problem and outperforms pipelined approaches (ER plus hierarchical clustering, in either order) in the general hierarchical ER setting.

Both `Hier-Type` and `HierER` leverage prior information such as the similarity of records based on image and text features. Even though computed similarity values can be noisy (e.g., records referring to different entities can have more similar features than records referring to the same entity) we show that they can be used effectively to minimize oracle queries. We provide theoretical analysis of `Hier-Type` and `HierER` under two representative similarity noise models, that we refer to as *data error* and *processing error*. The former model assumes higher noise in certain records, to capture the possibility of missing or incorrect data. The latter assumes higher noise in certain record pairs, to capture possible errors in the similarity computation process. Under moderate noise, `Hier-Type` and `HierER` require $O(n \log n)$ oracle queries and achieve optimal progressive F-score with high probability. Experimental results show that `Hier-Type` and `HierER` can scale up to datasets with millions of records and outperform baseline solutions both in efficiency and effectiveness. Since asking $O(n \log n)$ queries to an expert or a crowd worker may not be feasible for million scale datasets, we consider a classifier trained using active learning to act as an oracle. Such oracles are trained with less than 1000 queries to the crowdworker. Finally, in order to allow the possibility that the oracle itself makes mistakes we also demonstrate empirically the robustness of our methods when used together with the *oracle error* correction method from prior literature [26].

Outline. We formalize the problem statement in Section 2 and present our solution overview in Section 3. The details of our algorithm are presented in Sections 4-6, which is analyzed theoretically in Section 7. Section 8 discusses the related work and Section 9 evaluates our technique empirically with other strategies.

2 PROBLEM DEFINITION

In this section, we define the problem statement formally.

Notation. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a collection of n records and $T = \{t_1, \ldots, t_k\}$ be the initial set of type nodes organized in the form of a hierarchy H_T. We assume that each type in the seed hierarchy H_T contains at least one leaf-level record as an example[1]. Let H^* denote the ground truth of our hierarchical ER problem, i.e., a rooted tree containing the type-subtype relationships (including the type nodes T), is-A relationships (mapping each entity to a type) and co-reference (i.e., referring to the same entity, or equality) relationships and consisting of n leaves, each corresponding to a distinct record $v_i \in V$. H^* consists of three types of nodes i) records V, which are at the leaf-level ii) *type nodes* $T' \supseteq T$, and iii) *entity nodes* E. All children of an entity node in the hierarchy H^* refer to the same entity, while a type node consists of a set of entities (and hence of records). For simplicity of exposition, an entity can also be thought of as a type consisting of a singleton entity. In this way, all internal nodes of H^* can be thought of as connected by type-subtype relationships. We use the notion of *laminar* family of sets to define the hierarchy H^* more formally.

Definition 2.1 (Laminar Family). A family of sets $\mathcal{C}^*$ is laminar iff $\forall X_1, X_2 \in \mathcal{C}^*$, either $X_1 \cap X_2 = \phi$ or $X_1 \subseteq X_2$ or $X_2 \subseteq X_1$.

The hierarchy H^* corresponds to a laminar family of labelled sets $\mathcal{C}^*$ such that each set in $\mathcal{C}^*$ is labelled with one of the three labels: `record (r)`, `entity (e)` or `type (t)`. Each labelled set is denoted as $\langle \texttt{label} : X \rangle$ where $X \subseteq V$. According to this notation, $\langle \texttt{r} : \{v\} \rangle \in \mathcal{C}^*, \forall v \in V$ and $\langle \texttt{t} : V \rangle \in \mathcal{C}^*$. This hierarchy has an additional constraint that a set labelled '`entity`' cannot have a proper superset of label '`entity`' or '`record`' and a set labelled '`type`' cannot have a superset labelled '`entity`' or '`record`'.

Following this definition, there exists a one-to-one mapping between internal nodes of the hierarchy and the laminar family of sets $\mathcal{C}^*$ where an internal node of the hierarchy (say u) is equivalent to a set $C \in \mathcal{C}^*$ containing all the leaf-level descendants of u and vice versa. In this formulation, we assume that all type nodes can not have a single type node as a child in the hierarchy. A type node that has a single type node as a child is redundant and can be ignored. However, an entity can have a single entity reference (record) and hence have only one child.

Example 2.2. The hierarchy in Figure 3 is equivalent to a laminar family of 38 labelled sets where 14 sets are labelled `record`, 12 are

[1]This assumption is needed as our framework does not compare a type node with records. Our framework generalizes to the case when H_T does not contain records if pairwise queries allow comparing type nodes with records.

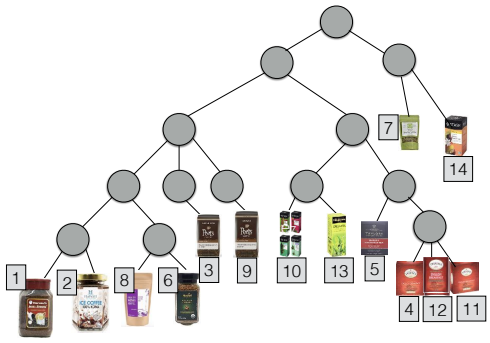


Figure 4: Constructed Hierarchy with triplet queries only

labelled `entity` (corresponding to e_1-e_{12}) and 12 are labelled `type` (u_1-u_{12}). Some of these sets are listed below.

$$\{\langle r:\{1\}\rangle \cup \langle r:\{2\}\rangle \cup \langle e:\{1\}\rangle \cup \langle e:\{2\}\rangle \cup \langle t:\{1,2\}\rangle \cup \langle t:\{1,2,6,8\}\rangle\}$$

The Lowest Common Ancestor (`lca`) of records v_1, v_2 is the common ancestor of both v_1 and v_2 that is farthest from the root. If the `lca` is an entity node, then v_1 and v_2 refer to the same entity.

Definition 2.3 (Depth). The `depth` of a node u is defined as the number of edges from the root to u. The root node has `depth` 0.

A hierarchy has an interesting property that for any three records $v_1, v_2, v_3 \in V$, the `lca`'s of two pairs of these records are the same and the `lca` of the third pair is either the same or a descendant of the other two `lca`'s [21]. Without loss of generality, one of the following hold.

$$\texttt{lca}(v_2, v_3) \geq \texttt{lca}(v_1, v_2) = \texttt{lca}(v_1, v_3)$$

where $\texttt{lca}(v_1, v_2) > \texttt{lca}(x, y)$ denotes that $\texttt{lca}(v_1, v_2)$ is a descendant of $\texttt{lca}(x, y)$. In case all three `lca`'s are the same, then v_1, v_2 and v_3 belong to three different descendant-branches of the internal node corresponding to $\texttt{lca}(v_1, v_2) = \texttt{lca}(v_1, v_3) = \texttt{lca}(v_2, v_3)$ [21].

Example 2.4. Consider the records 1, 4, 12 in Figure 3. We observe that $\texttt{lca}(4, 12) = e_{10}$ is present deeper in hierarchy as compared to $\texttt{lca}(1, 12) = u_7$, i.e., $\texttt{lca}(4, 12) > \texttt{lca}(1, 12)$ and $\texttt{lca}(1, 4) = \texttt{lca}(1, 12)$. For the records $(1, 3, 9)$, we observe that all pairwise `lca`'s are same, and thus the three records are present in different descendant branches of $\texttt{lca}(1, 3) = u_4$.

Oracle Abstraction. Consider a black box that outputs the relative arrangement of any triplet of records in the form of a hierarchy.

Definition 2.5 (Triplet Oracle). A triplet oracle is a function $q_t : V \times V \times V \rightarrow V \cup \{\phi\}$ that takes three records as input and outputs the farthest record (if any). For an input (v_1, v_2, v_3), the oracle outputs $q_t(v_1, v_2, v_3) = v_1$ if $\texttt{lca}(v_2, v_3) > \texttt{lca}(v_1, v_2) = \texttt{lca}(v_1, v_3)$ and $q_t(v_1, v_2, v_3) = \phi$ if $\texttt{lca}(v_1, v_2) = \texttt{lca}(v_1, v_3) = \texttt{lca}(v_2, v_3)$.

The oracle output is similar to comparing record pair similarities and returning a record that has the least similarity with the other two records (similar to an odd-one-out). A pair of records (v_2, v_3) is considered more similar than a pair (v_1, v_2), if (v_2, v_3) share more types in common than (v_1, v_2).

Example 2.6. Consider a triplet query with records 1, 2, and 3 in Figure 2. Record 3 is considered farthest from 1 and 2 as record pair $(1, 2)$ share five common types, which include beverages (u_{11}), caffeinated beverages (u_7), coffee (u_4), instant coffee (u_3), and spray fried instant coffee (u_1). In contrast, record pairs $(1, 3)$ and $(2, 3)$ share only three of these common types (u_{11}, u_7, and u_4). In other words, the pair $(1, 2)$ is more similar than the pair $(1, 3)$ and the pair $(2, 3)$ in terms of their hierarchical similarity (depth of their lowest common ancestor).

The result of $q_t(v_1, v_2, v_3)$ is the same across any ordering of the records v_1, v_2 and v_3, e.g., $q_t(v_1, v_2, v_3) = q_t(v_3, v_1, v_2)$. The information provided by a triplet query can be combined with the evidence from other triplet queries to generate the hierarchy over V. However, if we ask only triplet queries, we cannot identify entity nodes and our best result can be as in Figure 4, that is, without entity nodes $e_1, \ldots, e_{12}$ and with misinterpretation of e_{10} as a type node. To correctly identify all entity nodes, we consider an oracle that outputs whether any pair of records refers to the same entity [24, 65, 66].

Definition 2.7 (Equality Oracle). An equality oracle is a function $q_e : V \times V \rightarrow \{T, F\}$ that takes two records as input and outputs $q_e(v_1, v_2) = T$ (true) whenever u and v refer to the same entity, and $q_e(v_1, v_2) = F$ (false) otherwise.

Example 2.8. In Figure 2, records 4, 11, and 12 refer to the same entity. Therefore, the equality oracle returns T for $q_e(4, 12)$, $q_e(4, 11)$, and $q_e(11, 12)$ and F for other pairwise equality queries.

A recent work [21] showed that any strategy that queries all $\binom{n}{3}$ record triples can recover the underlying type-subtype hierarchy uniquely assuming that the triplet oracle makes no mistakes.[2] Since the maximum number of equality queries possible are $O(n^2)$, the $\Theta(n^3)$ upper-bound in [21] also holds for hierarchical ER. Equality and triplet oracles can be considered as an abstraction of a machine learning classifier or a domain expert that provides high-quality labels to an input query. We discuss practical implementation of these oracles in Section 9.

Progressive F-score. A naive way to evaluate performance of different hierarchical ER strategies is to compare the fraction of the total $\Theta(n^3)$ relationships (i.e., $\binom{n}{3}$ triplets and $\binom{n}{2}$ equality relationships) that are correctly identified by each of the strategies. However, such an approach can be infeasible even when n is as small as 10,000 (medium-sized datasets). Therefore, we extend the popular metric of comparing *F-score* of different ER techniques to our hierarchical setting. In ER, F-score is computed over two types of pairwise relationships: intra-cluster and inter-cluster. Following these ideas, we consider co-reference relationships as intra-cluster and enumerate the different types of inter-entity relationships between record pairs. We define the notion of `t-ancestor` relationship to capture the distance between record pairs and then use it to compute the F-score of the output hierarchy H.[3]

Definition 2.9 (`t-ancestor` relationship). A pair of records (u, v) satisfies a `t-ancestor` relationship if their `lca` is at most `t` edges

[2] [21] also showed that $O(n \log n)$ queries are sufficient for binary hierarchies.
[3] We remind the reader that F-score in the traditional ER setting is defined as the harmonic mean of precision and recall, and that precision and recall are defined with respect to the number of correctly identified equality relationships.

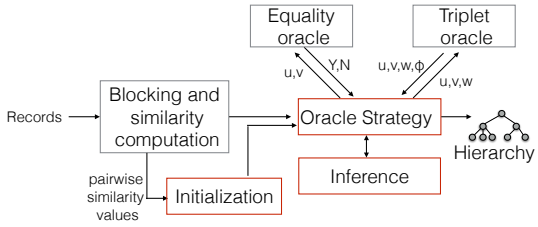


Figure 5: Our hierarchical ER workflow. The blocks outlined in red represent our novel contributions, whereas the blocking and similarity computation is implemented with the methods available in the literature.

away from both u and v's entity nodes. 0-`ancestor` relationship is equivalent to an equality (i.e., co-reference) relationship.

For example, record pair $(1, 14)$ satisfies a 5-ancestor relationship because entity node of 1 is five edges away from the `lca` (u_{11}) and entity node of 14 is two edges away from the `lca`. However, the record pair $(1, 14)$ does not satisfy any $t-$ancestor relationship for $t < 5$. We map t-`ancestor` relationships identified from the output hierarchy H to the ground truth hierarchy H^* and then use those to compute precision as the weighted fraction of correctly identified pairwise relationships among the identified relationships and recall as the weighted fraction of total relationships that were identified. We discuss the weighting mechanisms in Section 9. F-score is finally computed as the harmonic mean of precision and recall.

Finally, since evaluating F-score alone at the end of the process cannot distinguish between strategies that achieve higher F-score early on and strategies that achieve it only at the end, we also use *progressive F-score* to compare strategies. Progressive F-score corresponds to the area under the F-score vs. query sequence curve, where higher area corresponds to better performance [24, 26].

Problem statement. Now, we define our problem statement of developing a strategy that adaptively identifies oracle queries based on previously asked queries and pairwise record similarities.

Problem 1 (Hierarchical ER). *Given a collection of records V, initial seed hierarchy H_T, oracle access to H^* and a similarity function $s : V \times V \rightarrow [0, 1]$, find an adaptive querying strategy that maximizes the progressive F-measure of the recovered hierarchy.*

Problem 1 assumes that the oracle answers every query correctly. However, in practice humans make mistakes and to deal with noise in oracle response, we leverage prior techniques from ER literature [26]. Pairwise similarity values can be calculated using textual descriptions and image features. Finding the best similarity function is outside the scope of this work, but it is important to note that random or adversarial similarities would not be helpful to optimize the oracle queries. We provide theoretical analysis of our method for different noise models and use well-known similarity functions in our experiments to demonstrate its robustness.

3 OVERVIEW

We present an overview of our workflow for the hierarchical ER problem in Figure 5. Modules are described below.

Blocking and Similarity. To reduce the number of pairs considered for similarity computation, we perform *blocking* [53]. Blocking is a widely used operation in ER literature to efficiently generate a small set of candidate pairs so that similarity values are computed only for this small set of candidates. Standard blocking (also known as token-based blocking) is one of the most popular mechanisms that generates a block for each token in the input set of records [53]. Similarity values may not be directly interpretable as probability distributions over the possible oracle responses. For practical purposes, calibration approach in [24, 69] can be used to map values $s(v_1, v_2)$ to probability distributions $p(v_1, v_2)$ and $p(v_1, v_2, v_3)$, for the equality and triplet oracles respectively. Details about the blocking method and similarity functions used in our experiments are provided in Section 9.

Example 3.1. Consider the records from Figure 2. The different blocks identified by standard blocking correspond to individual tokens like a block for token 'Peet' contains records 3 and 9. The different blocks are ranked based on TF-IDF scores and low scoring blocks are dropped. Blocks corresponding to frequent tokens like 'tea', 'coffee' have a low score and are ignored for candidate pair enumeration. Therefore, record pairs $(1, 3)$, $(1, 14)$ are not identified as candidates because these record pairs do not co-occur in any of the high-scoring blocks. The identified candidate record pairs are used by subsequent stages for hierarchy construction.

Initialization. The initialization module in our workflow constructs a candidate hierarchy $\bar{H}$ that can be used downstream to guide the querying strategies. Construction of $\bar{H}$ is based solely on the similarity scores $s : V \times V \rightarrow [0, 1]$ and requires no oracle queries. We provide detailed discussion in Section 4 and prove that $\bar{H}$ has high F-score under low noise of similarity values (Section 7).

Inference. The inference module in our workflow provides tools to *infer* relationships from previously asked triplet and equality queries, without asking new oracle queries. Note that inferring type-subtype relationships is the major challenge in this module, as it is known from previous ER literature [65, 66] that equality relationships can be easily inferred via transitive closure.[4] We provide detailed discussion in Section 5.

Oracle strategy. This module has the goal of prioritizing oracle queries by leveraging (i) the candidate hierarchy from the initialization step to give higher priority to queries that yield higher F-score increment and (ii) the inference engine to identify inferable relationships for free. We provide implementations of `Hier-Type` and `HierER` approaches in Section 6.

`HierER` strategies. Given the workflow in Figure 5, `Hier-Type` and `HierER` can be thought of as oracle strategies that leverage the initialization and inference methods and focus on the following principles to maximize progressive F-score.

• *Internal node discovery.* `Hier-Type` and `HierER` prioritize queries that enable the discovery of new internal nodes. Indeed, identifying the tree structure, specifically the internal nodes between the root node and the leaf nodes corresponding to the processed records is important to provide optimal progressive behavior.

• *Large entities.* `Hier-Type` and `HierER` give high priority to queries enabling the discovery of new children of high-degree entity nodes. This principle has also been used in previous ER

[4] E.g., if we know that v_1 refers to the same entity as v_2, and v_2 refers to the same entity as v_3, then we can infer that v_1 refers to the same entity as v_3 without asking the corresponding oracle query.

Algorithm 1 `generate-similarity-hierarchy`

Require: records X, similarity s, degree α
1: **if** $|X| == 1$ **then return** X
2: $\bar{H}_X \leftarrow$ `new-node()`
3: **if** $|X| \leq \log n$ **then**
4: $\quad \mathcal{P} \leftarrow$ `greedy-partition`(X, s, α)
5: **else**
6: $\quad \mathcal{P} \leftarrow$ `robust-partition`(X, s, α)
7: **for** $C \in \mathcal{P}$ **do**
8: $\quad \bar{H}_c \leftarrow$ `generate-similarity-hierarchy`(C)
9: $\quad \bar{H}_X$.`add-children`$(\bar{H}_C)$
10: $\bar{H}_X \leftarrow$ `merge-internal`$(\bar{H}_X)$
11: **return** $\bar{H}_X$

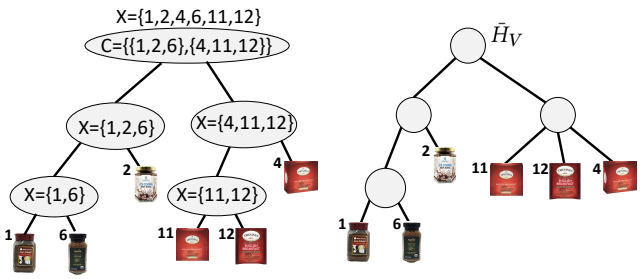


Figure 6: Example candidate hierarchy (with $\alpha = 2$) where the internal node corresponding to $\{4, 11, 12\}$ is merged with its child $\{11, 12\}$ by `merge-internal`. The hierarchy incorrectly places $(1, 6)$ closer than $(1, 2)$ due to noise in similarity values.

literature [24, 65] since asking queries in non-increasing order of entity sizes provides the maximum gain in progressive recall.

• *Connectivity.* `Hier-Type` and `HierER` prioritize queries that grow the hierarchy in a connected fashion. Indeed, ensuring that the processed records at any given time form a single connected hierarchy (rather than growing multiple disjoint hierarchies in parallel) allows for the inference of more relationships with the same number of queries.

4 INITIALIZATION

We now present a top-down algorithm (Algorithm 1) to construct a candidate hierarchy $\bar{H}_V$ capturing type relationships from the similarity values $s : V \times V \rightarrow [0, 1]$.

Algorithm 1 takes a collection of records $X \subseteq V$ as input along with pairwise similarities s and an optional parameter α (default value $\alpha = 2$) denoting the minimum degree of `type` labelled nodes in the ground truth hierarchy. It outputs a laminar representation of a hierarchy $\bar{H}_X$ containing records in X at the leaf-level. If X is a singleton set, it is returned as the leaf level of the hierarchy (line 1). Whenever $|X| > 1$, $\bar{H}_X$ is initialized with a new node (line 2) as the root of the hierarchy (which is equivalent to X in the laminar representation). X is then partitioned into α clusters (denoted by $\mathcal{P}$) by `greedy-partition` if $|X| \leq \log n$ and `robust-partition` otherwise (lines 3-6). The partitions $\mathcal{P}$ capture the different children branches of the root $\bar{H}_X$. The candidate hierarchy on each partition $C \in \mathcal{P}$ is identified by recursively invoking Algorithm 1 and the root node of these branches $\bar{H}_C$ is added as a child to the root node $\bar{H}_X$ (lines 7-9). Some internal nodes in the ground truth hierarchy may have degree more than α. To identify such nodes, `merge-internal` post-processes $\bar{H}_X$ to consolidate internal nodes that have higher likelihood of referring to the same node. This procedure processes

Algorithm 2 robust-partition

Require: records X, similarity s, degree α
1: Select subsets $S, S' \subseteq X$ randomly such that $|S| = |S'| = \Theta(\log n)$
2: **for** $v_1 \in S$ **do**
3: $\quad$ **for** $v_2 \in S \setminus \{v_1\}$ **do**
4: $\quad\quad Count(v_1, v_2) \leftarrow \sum_{v_3 \in S'} \mathbb{1}\{s(v_1, v_2) > s(v_1, v_3), s(v_2, v_3)\}$
5: $\quad\quad$ **if** $Count(v_1, v_2) > |S'|/(2\alpha)$ **then**
6: $\quad\quad\quad A^+ \leftarrow A^+ \cup \{(v_1, v_2)\}$
7: $\quad\quad$ **else**
8: $\quad\quad\quad A^- \leftarrow A^- \cup \{(v_1, v_2)\}$
9: $C \leftarrow \phi$
10: **while** $|C| < \alpha$ **do**
11: $\quad C' \leftarrow$ select a random record from S (say v_1)
12: $\quad$ **for** $v_2 \in S$ **do**
13: $\quad\quad$ `ACount`$(v_1, v_2) = \sum_{v_3 \in S} \mathbb{1}\{(v_1, v_3)$ and $(v_2, v_3) \in A^+$ or $A^-\}$
14: $\quad\quad$ **if** `ACount`$(v_1, v_2) > |S|/2$ **then**
15: $\quad\quad\quad C' \leftarrow C' \cup \{v_2\}$
16: $\quad S \leftarrow S \setminus C', C \leftarrow C \cup C'$
17: $\beta \leftarrow \min_{C_i \in C} |C_i|$
18: **for** $v_3 \in X \setminus \cup_{C_i \in C} C_i$ **do**
19: $\quad$ `assign-count`$(v_3, C_i) = \sum_{t \leq \beta} \mathbb{1}\{s(v_3, C_i[t]) > s(v_3, C_j[t]), \forall j \neq i\}$
20: $\quad C_\eta \leftarrow \arg\max_{C_i \in C}$ `assign-count`(v_3, C_i)
21: $\quad C_\eta \leftarrow C_\eta \cup \{v_3\}$
22: **return** C

parent-child relationships in bottom-up manner and integrates two internal nodes if the probability of a merged node is higher than the probability of split internal nodes. Such probabilities are estimated by considering triplets with these internal nodes as `lca` and their calibrated probability distribution[5].

`greedy-partition`. is a greedy algorithm to identify the different branches of the set X. This algorithm is same as the greedy k-center algorithm [31] where each branch is initialized by a seed node and remaining nodes are assigned to the closest seed nodes. Given a set of records X, it initializes the first branch by randomly choosing a leaf level record as a seed and all records in X are assigned to the first seed. The subsequent seed nodes are identified by choosing the record that has minimum similarity from its assigned seed record. All records in X are then re-assigned to the identified branches where each record is assigned to the closest seed node. Notice that the identified minimum similarity record may be inaccurate due to noise in similarity values which can lead to noisy hierarchy construction. To improve the robustness of the partitioning algorithm, we propose `robust-partition` subroutine that is proven to be accurate when the different branches contain more than $\log n$ records.

Example 4.1. Consider the records V from Figure 2. Figure 6 shows the recursive tree for $X = \{1, 2, 4, 6, 11, 12\} \subset V$ constructed by greedy partition. In the first iteration, the greedy partition algorithm initializes the first branch with record 2 and then chooses record 11 for the second branch as it has the lowest similarity with 2. After identifying these seed nodes, all other records are assigned to these branches. Therefore, 1, 6 are assigned to the branch containing 2 and others are assigned to 11. In the subsequent stages, $\{1, 6, 2\}$ is partitioned to form the left sub-tree and $\{4, 11, 12\}$ is partitioned to form the right sub-tree. The internal node $\{11, 12\}$ is merged with its parent by `merge-internal` due to the high-probability of referring to the same entity. Note that this hierarchy is not accurate

[5]The calibration of similarity values is performed in practice to estimate probability values. We leverage prior techniques for this [24, 69].

due to the noise in similarities (image similarity of $(1, 6)$ is higher than that of $(1, 2)$).

robust-partition. This mechanism is a robust adaptation of greedy−partition which considers similarity of multiple record pairs during branch assignment. Algorithm 2 presents the pseudocode which operates in three main steps. In the first step, we consider two samples of $\Theta(\log n)$ records (say S and S'). Every record pair $(v_1, v_2) \in S \times S$ is assigned a score equal to the number of triangles formed by (v_1, v_2) with nodes in S' such that $s(v_1, v_2)$ is greater than the similarity of other edges of the triangle (line 4). Using these count values, each edge is labelled intra-branch (A^+) or inter-branch (A^-). In the absence of noise, edge labels A^+ and A^- are accurate but due to noise in similarity values, the labels may be incorrect. To correct the labelling, we calculate agreement score (`ACount`) of each edge $(v_1, v_2) \in S \times S$ which is equal to the number of times, edges (v_1, v_3) and (v_2, v_3) have same labels for different $v_3 \in S$ (line 13). The nodes in S are partitioned into α clusters by thresholding the agreement score of edges (lines 10-16). We later show that the labels identified by this two-step procedure are robust to noise in similarity. Once a partitioning of S is identified, all the remaining records in X are assigned to one of the partitions by comparing their similarity to different points in these partitions (using `assign-count`, lines 19-21).

5 INFERENCE

We now provide a method to *infer* relationships from previously asked triplet and equality queries, without asking new oracle queries. A given set of queries $Q \subset V \times V \times V$ may not be enough to arrange all involved records in the form of a unique hierarchy. However, we can construct a collection of hierarchies such that all triplet relationships that are either queried or can be inferred[6] from Q can be answered from one of the constructed hierarchies. To better understand this behavior, we mathematically characterize the information available from the following example triplet queries. We use the notation $\texttt{lca}(x, y) > \texttt{lca}(u, v)$ to denote that $\texttt{lca}(x, y)$ is a descendant of $\texttt{lca}(u, v)$ in the hierarchy.

Consider a query $q_1 \equiv q_t(1, 2, 6)$ that returns 6. It is equivalent to a hierarchy consisting of three leaf-level records. To interpret q_1 mathematically, we define three variables corresponding to the lca's of involved record pairs $(1, 2)$, $(1, 6)$ and $(2, 6)$. Using these variables, q_1 can be represented as follows.

```
lca(1,2) > lca(1,6) = lca(2,6)
```

This inequality characterizes a relation between the lca of record pairs $(1, 2)$, $(1, 6)$ and $(2, 6)$. Each query can be written in the form of such inequality constraints over at most $\binom{n}{2}$ lca variables. Consider another query $q_2 \equiv q_t(1, 6, 11) = 11$,

```
lca(1,6) > lca(1,11) = lca(6,11)
```

Using the inequalities of q_1 and q_2, we can infer that

```
lca(1,2) > lca(1,6) > lca(1,11)
```

With this evidence we can place the four records 1, 2, 6 and 11 on the same branch (Figure 7(a)). This example describes the merge operation over two hierarchies where the node 11 is inserted into the first hierarchy based on response to query q_2. However, if the query q_2 is not asked and instead $q_3 \equiv q_t(1, 2, 11)$ returns 11 then,

```
lca(1,2) > lca(1,11) = lca(2,11)
```

In this case we cannot infer if $\texttt{lca}(1, 6) > \texttt{lca}(1, 11)$. Therefore, we cannot merge the hierarchies corresponding to q_1 and q_3 at the moment, but more queries like $(1, 6, 11)$ will get enough evidence to merge the hierarchies following the same procedure. Following this notation, an equality query of the form $q_4 \equiv q_e(u, v) = T$ reveals that u and v refer to the same entity and is the same as merging the nodes corresponding to u and v in $\mathcal{H}$. All such relationships can be inferred via transitive closure (see [65, 66] for more details). Using these properties, we now present an approach to maintain the collection of hierarchies $\mathcal{H}$ over the queries Q where a node may be present in multiple hierarchies and all equivalent copies of a node are connected by equivalence edges. For example, lca of records v_1 and v_2 in one hierarchy is equivalent to lca of v_1 and v_2 in another hierarchy. The collection of hierarchies $\mathcal{H}$ is sequentially updated with the addition of each query to Q.

The inference algorithm considers a collection of hierarchies $\mathcal{H}$ and a new query $q \equiv q_t(x, y, z)$ as input and outputs an updated collection of hierarchies. First, a new hierarchy H consisting of three leaf level records x, y, z is inserted into the collection $\mathcal{H}$ and a list of modified hierarchies is initialized with H. This list contains all the hierarchies that have been modified due to the insertion of q. Additionally, we maintain a mapping of hierarchies to a set of records that have been newly added into the hierarchy (`newly-added`(H)) since it was last processed by the inference algorithm. The algorithm then tries to merge each of the modified hierarchies with other hierarchies already present in $\mathcal{H}$ (following the procedure described above). The merge operation then returns a new list of modified hierarchies which are considered for subsequent iterations of `merge` and the process continues until all have been processed.

6 ORACLE STRATEGIES

In this section, we present oracle querying strategies to solve Problem 1. First, we present auxiliary methods and then use them to discuss `HierER` algorithms that optimize progressiveness.

6.1 Auxiliary Methods

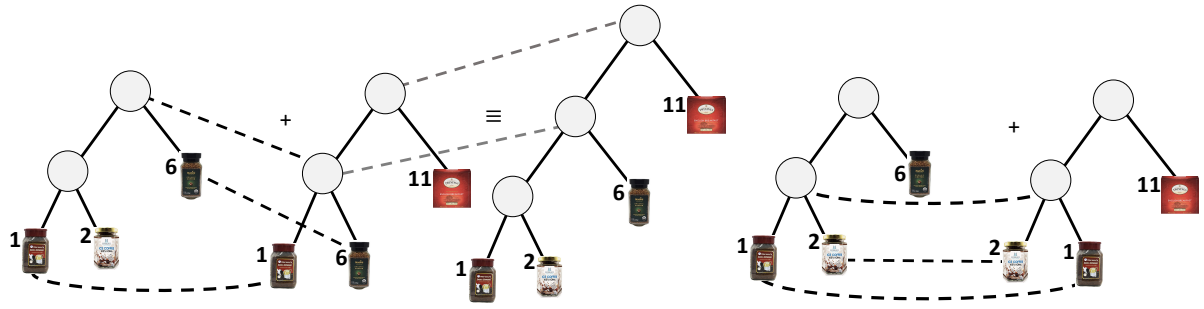
We define subroutines that are used by the oracle strategies.

identify-branch. Our first auxiliary method takes as input a leaf level record v and an internal node u of the hierarchy H that does not contain v yet and returns the branch of u where v should be inserted. This helps to identify relative position of v with respect to u and narrow down the search for the exact location to insert v.

Example 6.1. Consider a hierarchy H (as shown in Figure 7(a)) over the records $\{1, 2, 6, 11\}$. Running `identify-branch` with record $v = 8$ and internal node u denoting the root node of H, would yield the left branch as the output because 8 should be added to the left branch of u according to the ground truth.

It considers all children branches of u as candidates to search for v's location and iteratively queries a triplet containing v and two leaf-level records (l_1 and l_2) belonging to different candidate branches (lines 4-5). The pair (l_1, l_2) has the property that $\texttt{lca}(l_1, l_2) =$

[6] A triplet is considered inferrable, if its oracle response can be inferred from one of the prior queries.



(a) Two queries can merge. (b) Can not infer new relationships.

Figure 7: Collection of hierarchies for two different sets of queries, where the dashed lines denote equivalence edges and solid lines capture ancestor-descendant relationships.

u. If a query (v, l_1, l_2) returns l_1 then $\texttt{lca}(v, l_2) > \texttt{lca}(l_1, l_2)$, implying v belongs to the same branch as l_2. Similarly, the respective branch is also identified if the oracle returns l_2 or v. However, if the query returns ϕ, then v does not belong to the branches corresponding to l_1 and l_2. In this case, the querying procedure continues with other pairs of branches. If the response to first t queries is ϕ, v is attached as a separate branch to u. t is used as an early stopping threshold to not explore all children branches. If $t \geq \lceil |C|/2 \rceil$ and the algorithm returns the addition of a new branch then v is guaranteed to belong to the subtree rooted at u. We also consider an extension of this method, such that if a pair of internal nodes $(u, u_1) \in H$ is given as input, where u_1 is a child of u, then u_1 is considered as the first candidate in the list C containing $u.\texttt{children}()$ to check if v is present in the same branch as u_1. If v is seen to be present in this branch then $\texttt{identify-branch}(v, u_1, t)$ is used to return the branch of v. This extension is particularly useful if our techniques expect v to be inserted between (u, u_1).

Find Sibling. The `find-sibling` subroutine takes a leaf-level record v as input and identifies a sibling of v in a hierarchy H. `find-sibling` queries v with an internal node u (using `identify-branch` subroutine) to identify the relative position (branch) of v with respect to u. The node u is chosen such that the maximum size of the components formed by removing u and its edges from H is minimized. Following the response of $\texttt{identify-branch}(u, v)$ it identifies the partition that v belongs to and thereby reduces the search space with the `update` procedure (line 5). This recursive procedure stops when a single element is left in the set of candidates. This algorithm is similar to binary search and requires $O(\alpha \log n)$ queries where internal nodes have degree less than α.

Example 6.2. Consider a hierarchy H (as shown in Figure 7(a)) over the records $\{1, 2, 6, 11\}$. Running `find-sibling` with record $v = 8$ would yield the record 6 as its sibling by recursively running `identify-branch` subroutine.

Calculate Benefit. The `benefit` of a record v characterizes the gain in recall if v is inserted into the hierarchy. Insertion of v can lead to one of the following two cases. (a) v is attached to an existing internal node in the hierarchy (as a new branch) (b) insertion of v discovers a new internal node to which v is attached. `benefit-triplet` calculates the benefit of attaching v to an internal node, say u, which is defined as the expected gain in recall per query if v is attached to u. Mathematically, $\texttt{benefit}(u, v) = p_{u \to v} \times recall_g(u \to v)$ where $p_{u \to v}$ denotes the probability that v is attached to u and $recall_g(u \to v)$ is the gain in recall per query.

The gain in recall per query is measured in terms of the new relationships identified after inserting v divided by the number of queries required to attach v to u. For triplet queries, $p_{u \to v}$ is estimated from $\texttt{identify-branch}(v, u)$ with a difference that the similarity values are used to estimate the likelihood instead of oracle queries. Using this benefit calculation mechanism, `benefit-triplet` subroutine sorts all internal locations of the hierarchy H in non-increasing order of their benefit. Similarly, `benefit-equality` reuses the benefit calculation method in [24] to sort all the entity nodes in decreasing order of `benefit` for pairwise queries. This method returns top-τ entity nodes that have high benefit (where $\tau = \log n$). We use the same parameters as [24]. We present the pseudocode of `benefit-triplet` and `benefit-equality` in the full version [25].

6.2 HierER algorithms

Algorithm 3 presents the pseudocode for `Hier-Type` that generates a hierarchical arrangement of records capturing type-subtype relationships and ignores equality relationships. It is initialized with a candidate hierarchy ($\bar{H}$) generated by Algorithm 1 and the records V are sorted in non-increasing order of expected cluster size, where expected cluster size of a record v, $E(v) = \sum_u s(u, v)$ [24]. The algorithm then proceeds in two phases. The first phase (lines 2-5) iteratively inserts records into the processed hierarchy H (initialized as the seed hierarchy H_T) to achieve high progressive benefit by growing large clusters. This phase is initialized with a set of processed records P, a set of tentatively processed records P_t and a hierarchy H over $P \cup P_t$. The records V are sequentially inserted into H in non-increasing expected cluster size using `Hier-TypeInsertion`. At the end of this phase, all the records are placed in the form of a hierarchy H but the exact location of the records P_t may not have been verified, due to the early stopping criterion of `identify-branch` to attach a new branch. Such unverified nodes are re-processed in the second phase using `find-sibling` to find their exact location in the hierarchy. We now describe `Hier-TypeInsertion` in detail.

`Hier-TypeInsertion` phase. Algorithm 4 takes a leaf-level record v, hierarchies H and $\bar{H}$, sets of processed records P and P_t as input and outputs the updated H and the processed nodes. It first sorts all the locations in H in non-increasing order of benefit and maintains a list S of these candidates. Each internal node is queried sequentially in this sorted order with an additional constraint that all internal nodes discarded by previous queries are not considered

Algorithm 3 `Hier-Type`

Require: records V, seed hierarchy H_T, similarity s
1: $\bar{H} \leftarrow$ `generate-similarity-hierarchy`(V)
2: $L \leftarrow$ `get-expected-cluster-size`(V)
3: $P \leftarrow$ `leaves`$(H_T), P_t \leftarrow \phi, H \leftarrow H_T$
4: **for** $v \in L$ **do**
5: $\quad H, P, P_t \leftarrow$ `Hier-TypeInsertion`$(v, H, P, P_t, \bar{H})$
6: **for** $v \in P_t$ **do**
7: $\quad root \leftarrow u$ such that $u \in H \setminus P_t, u = v.ancestor()$
8: $\quad H \leftarrow$ `find-sibling`$(v, root)$
9: **return** H

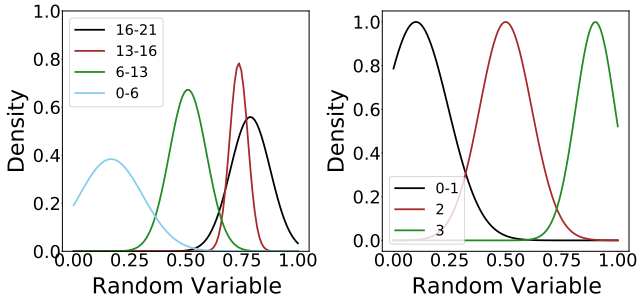
Algorithm 4 `Hier-TypeInsertion`

Require: record v, Hierarchy H, Processed records P, P_t, candidate hierarchy $\bar{H}$
1: $B \leftarrow$ `benefit-triplet`(v, H, P)
2: $i \leftarrow 1$
3: $S \leftarrow initialize - candidates(H)$
4: **for** $b \in B \cap S$ **do**
5: $\quad o, t \leftarrow$ `identify-branch`$(v, b, \tau - i)$
6: $\quad i \leftarrow i + t$
7: $\quad S \leftarrow update(S, o)$
8: $\quad$ **if** $|S| = 1$ **then**
9: $\quad\quad$ Insert b as a sibling of S
10: $\quad\quad$ break
11: $\quad$ **if** $i \geq \tau$ **then**
12: $\quad\quad$ Attach v to the candidate with lowest depth
13: $\quad\quad P_t \leftarrow P_t \cup \{v\}$
14: $\quad\quad$ **return** H, P, P_t
15: $\quad\quad$ break
16: $\quad$ **if** $i \geq \gamma$ **then**
17: $\quad\quad u \leftarrow$ `find-sibling`(v, S, H)
18: $\quad\quad$ break
19: $H \leftarrow$ `expand-branch`$(v, H, \bar{H})$
20: $P \leftarrow P \cup \{v\}$
21: **return** H, P, P_t

as candidates. The queries involving v are asked until one of the following three conditions is satisfied. (i) a unique location to insert v is identified (lines 8-10). (ii) the number of queries involving v exceeds γ, where γ refers to the estimate of queries required by `find-sibling` to insert v (lines 16-18). In this case, the benefit based querying is ignored and `find-sibling` subroutine is used to process v. (iii) the number of queries involving v exceeds τ (early stopping threshold, set to $\log n$). In this case, v is attached to the internal node that belongs to S, is present closest to the root node (lines 11-15), and v is marked tentative (line 13).

Once the record has been inserted, the `expand-branch` method identifies more internal nodes on the branch from the root node to v. It considers the branch from the root to v in the initialized hierarchy $\bar{H}$ to identify new internal nodes (say u) that have not been identified in H. It then selects the records v' such that $\texttt{lca}(v, v') = u$ in $\bar{H}$ and inserts v' into the hierarchy H.

Extension to `HierER`. `HierER` uses the equality oracle in addition to the triplet oracle to identify equivalence relationships. `HierER` proceeds similar to `Hier-Type` with a difference that it calculates the benefit of v with respect to triplet queries (same as `Hier-Type`) and equality queries (same as `Hybrid` [24]). The sorted list of all candidate locations to insert v are then sorted in non-increasing order of benefit. The record v is queried in non-increasing order of benefit, until it reaches one of the stopping conditions (same as the stopping criteria in `Hier-Type`). For an equality oracle query, the entity cluster for v is identified if q_e returns `True`. This algorithm



(a) `phyogenetic` (b) `camera`

Figure 8: Similarity distributions for pairs at different depths of lca.

is particularly beneficial when the dataset contains a large cluster of records referring to the same entity.

7 THEORETICAL ANALYSIS

We now analyze the query complexity and progressiveness of our algorithm under different similarity noise models, motivated by real world datasets. We show that `HierER` recovers the hierarchy in less than $O(n \log n)$ queries for most realistic settings. We present extensions of these results to analyze progressiveness and proofs in the full version [25].

We empirically observed that similarity of a pair of records is distributed normally with the mean varying based on the depth of $\texttt{lca}(u, v)$. This noise model is motivated by real world datasets, `phylogenetic` and `camera` (See Section 9 for more details). Figure 8 shows the similarity distribution of pairs at different depths (after curve fitting) in the hierarchy. More formally,

Definition 7.1 (Similarity distribution). The similarity of a pair of records (u, v), denoted by $s(u, v)$ with $\texttt{depth}(\texttt{lca}(u, v)) = d$, is sampled independently according to a normal distribution with mean μ_d and variance σ^2.

However, the similarity distribution of certain pairs is erroneous due to presence of noise in records. We study two different types of noise in pairwise similarities.

Processing error considers independent noise in similarities.

Definition 7.2 (Independent edge/processing error). Given a pair of records (u, v) such that $\texttt{depth}(\texttt{lca}(u, v)) = d$. The pairwise similarity $s(u, v)$ is distributed normally with mean μ_d and variance σ^2 with probability $1 - p$ and mean μ', where $|\mu' - \mu_d| \geq \epsilon$ with probability p.

The probability p captures the fraction of record pairs that have erroneous similarity values and ϵ captures the degree of error. We show the probability of attaching a leaf-level record v to an internal node x is estimated accurately with a high probability whenever x contains at least $16 \log n/(1 - 2p)^2$ leaf-level records in the subtree. Therefore, u is queried incorrectly with at most $16 \log n/(1 - 2p)^2$ nodes in the tree. Using this intuition, we show the following result.

Theorem 7.3. *For a collection of records V, Algorithm* `HierER` *constructs H^* in $O(n \log n/(1 - 2p)^2)$ with a probability of $1 - 1/n$ if $p < 0.5$ and $\sigma^2 < \mu/10$, where $\mu = \min\{\mu_d - \mu_{d'}\}$ for all d, d'.*

Edge-error model assumes that different pairs of records are noisy independently with probability p. However, pairwise similarities are dependent on features of records which are often noisy for

all records $u \in V$ that contain noisy tokens e.g. all herbal tea's are confused with caffeinated teas because of absence of token 'caffeine' from their descriptions. The systematic data error model captures such dependent noise between records.

Definition 7.4 (Data error). A leaf level record $u \in V$ has node error if pairwise similarity of all record pairs $(u, v)\ \forall\, v \in V \setminus \{u\}$ are sampled according to a normal distribution with mean μ' and variance σ^2 where μ' is the depth of $\texttt{lca}(u, v)$ in a noisy hierarchy $\mathcal{H}'$. $\mathcal{H}'$ denotes the modified hierarchy where u is attached arbitrarily to any internal node other than its true location in H.

To further generalize this noise model to define similar noise in a collection of nodes, we propose the systematic noise model.

Definition 7.5 (Systematic data error). An internal node u is considered to have systematic data error if the similarity value of q fraction of the leaf level records under u (say W) are noisy with data error. Among W, α fraction are connected arbitrarily in the noisy hierarchy $\mathcal{H}'$ and the remaining $1-\alpha$ fraction (say W_1) are present under the same internal node $x \neq u$, where $x \in \mathcal{H}$.

Under this noise model, we show that whenever u has more than $\frac{16}{(1-\alpha)q} \log n$ leaf level nodes, in expectation, initial $16 \log n$ of the erroneous set W_1 require at most $O(n)$ queries to be processed. However, remaining nodes in W_1 would be compared with only two internal nodes in the hierarchy, one of which is the true internal node. Therefore, all the remaining nodes in W_1 require at most $O(1)$ internal-node queries to identify the correct location in the hierarchy.

Theorem 7.6. *For a collection of records V, if an internal node u suffers from systematic data error with $q < 0.5$ and $\alpha = \Theta(\log n/n)$, then HierER algorithm constructs H^* in $O(n \log n)$ triplet comparisons with a probability of $1 - \frac{1}{n}$.*

8 RELATED WORK

We now discuss the prior techniques that are related to hierarchy construction or oracle based querying strategies.

Hierarchy Construction. The problem of recovering a hierarchy over a collection of records is closely related to taxonomy construction [7, 35, 45, 50, 51, 60], phylogeny construction [21], and hierarchical clustering [14, 16, 22]. Much of the prior work on taxonomy construction focuses on building a type hierarchy over textual data by first identifying a pairwise hypernym-hyponym relation between records and then cleaning the noise to induce a hierarchy. All these techniques are completely automated and do not leverage any supervision in the form of oracle queries, leading to sub-optimal quality [56]. Recent techniques [56] assume that all the internal nodes of the hierarchy are known a priori and a seed sub-hierarchy is given to initialize the pipeline. Prior hierarchical clustering techniques [14, 16, 22] use pairwise record similarities and do not leverage oracle queries to construct a binary hierarchy.

Phylogeny (Evolutionary trees) construction has been widely studied with the objective to organize the evolution of species in the form of a hierarchy [6, 9, 12, 32, 58]. Most of the internal nodes of a phylogeny have degree 2 with a few exceptions having degree 3. In this setting, a recent oracle based querying strategy [21] studied triplet querying strategies to generate binary hierarchies.

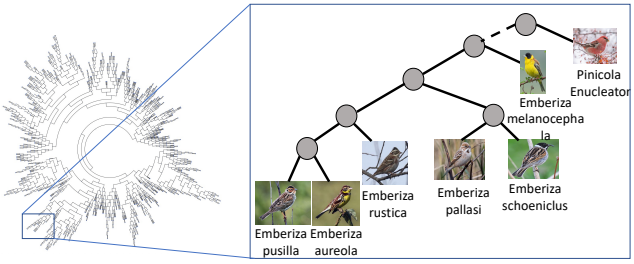


Figure 9: Example ground truth hierarchy structure for the `Phylogenetic` dataset. The center of each diagram corresponds to the root node and all the subsequent levels are placed in the form of concentric circles with the outermost circle corresponding to the leaf level nodes.

This algorithm uses `FindSibling` subroutine and guarantees hierarchy construction within $O(n \log n)$ query complexity for binary trees but may require $O(n^2)$ queries for non-binary hierarchies as it does not leverage pairwise similarities to guide the querying algorithm. Other hierarchical clustering techniques [29] optimize for a clustering objective to generate binary hierarchies.

Entity Resolution is one of the fundamental components of data integration which has evolved from rule based techniques [23] to learning based techniques and recently, crowdsourcing based techniques. There has been a lot of work on different aspects of ER (we refer the reader to [18, 28] for a more detailed survey on ER). In this work, we focus on crowdsourced ER techniques [24, 30, 33, 63–66]. These techniques consider access to machine generated pairwise probabilities, capturing their likelihood to refer to the same entity and an equality oracle. These techniques are helpful to identify parent of leaf level nodes in the hierarchy. Learning-based entity resolution techniques [20, 46, 49] focus on answering equality of record pairs, which is abstracted as an oracle in this work and can not be used as a strategy to solve problem 1.

Oracle based querying techniques. Prior work has extensively studied the use of comparison queries, which is a generalization of triplet queries. Comparison queries consider four records (say v_1, v_2, v_3, v_4) as input and compare the relative distance between (v_1, v_2) with that of (v_3, v_4). Such queries have been used to study correlation clustering [5, 59], classification [38, 57], top-k selection [13, 17, 19, 34, 43, 44, 54, 61], skyline computation [62] and many other machine learning tasks. Many empirical crowdsourcing studies have shown the ability of crowd members to answer such queries accurately [5]. An independent line of work [8, 68, 71] studies game theoretic mechanisms to decide task assignment among crowd workers. Such optimization is delegated to the implementation of an oracle and is orthogonal to our work.

9 EXPERIMENTS

In this section, we test the effectiveness of `HierER` on a diverse set of real-world datasets and answer the following questions. **Q1:** What is the end-to-end quality vs query complexity trade off for `HierER` as compared to other baselines? **Q2:** Is `HierER` sensitive to noise in the dataset? **Q3:** Is `HierER` scalable to large-scale datasets?

Table 1: Dataset description.

Dataset	n	Depth	Average degree	Max degree	Largest Entity
Phylogenetic	1039	21	2.01	4	1
DMOZ	100K	5	274.3	17K	1
Cars	16.5K	2	330	1800	1800
Camera	30K	3	5	91	91
Amazon	30K	7	8.46	760	1
Geography	3M	5	8K	35K	1

9.1 Setup

Before presenting the results, we describe the different datasets, experiment setup and the baselines considered in the evaluation.

Datasets. We consider six different real-world datasets consisting of hierarchies of varying depth, average degree and shape. Figure 9 shows the shape of one such hierarchy.

• `Phylogenetic` dataset contains scientific names of bird and insect species along with textual descriptions collected from Wikipedia. The hierarchy corresponds to the phylogenetic tree obtained from [55].

• `DMOZ` [1] is an open-content directory of web pages along with a hierarchy that organizes these webpages according to their categories like art, science, mathematics, etc. The dataset contains a dump of the text on the web page along with the ground truth.

• `Cars` comprises images of different models of cars. We generate textual descriptions using Google's vision API [2] and hierarchy is constructed based on their make and model.

• `Camera` [15] is a collection of specifications of cameras collected from over 25 retail companies and the hierarchy corresponds to the brand-model categorization.

• `Amazon` [36] contains descriptions of products and the amazon catalog ontology is used as the ground truth.

• `Geography` dataset [4] contains names of cities across the world and the internal nodes of the hierarchy correspond to their state, country and continent.

Experimental Setup. We implemented all techniques in Java and the code was run on a server with 64GB RAM. For blocking and pairwise similarity calculation, we considered tokens in the textual descriptions of the records. We performed meta-blocking [53] over the records, partitioned the edges of each record in the graph into $\log n$ buckets based on similarity and identified 100 triples per record from each bucket. This procedure ensures that $O(n \log n)$ total pairs of records are enumerated after blocking. For evaluation, we considered exponential decay in weights, i.e. weight of $(i+1)$-`ancestor` relationship is half the weight of i-`ancestor` relationship.

To calibrate pairwise similarity of records into triplet probability, we follow the procedure described in [24, 69]. We construct 100×100 buckets of equal width by considering a two-dimensional partitioning of similarities. A triplet (u, v, w) is mapped to a bucket corresponding to $(\max\{s(u,v), s(u,w), s(v,w)\}, \min\{s(u,v), s(u,w), s(v,w)\})$. We sampled $\log n$ samples from each bucket to calculate their probability mapping. See [24, 69] for more details.

Oracle implementation. Due to low-training data requirements for random forest classifiers as compared to deep-learning based techniques [49], we used a random forest classifier trained with active learning for `cars`, `geography` and `camera` datasets as an oracle. This procedure required less than 920 queries for `cars`, 560 for `camera` and 380 for `geography` dataset. The trained model achieved more than 0.95 F-score for all three datasets. For other datasets, we consider a simulated oracle model that used ground truth hierarchy to generate responses.

Baselines. To evaluate the effectiveness of our techniques, we consider the following pipelined baseline strategies that perform hierarchical clustering and entity resolution separately as a two-step procedure. (i) Average Linkage (denoted by `AverageLink`) is an Agglomerative clustering technique. We used the sklearn package [3] to construct the hierarchy and then run triplet queries bottom up to merge neighboring internal nodes. (ii) `HiExpan` denotes the automated taxonomy construction technique from [56] that uses the initial seed hierarchy to generate other internal nodes assuming access to all internal nodes in the hierachy. We added the internal nodes to run this algorithm. (iii) `InsSort` considers the extension of the insertion sort algorithm [21] for non-binary hierarchies to generate a hierarchy, followed by `hybrid` [24] to identify equality relationships. (iv) The pipeline that performs ER followed by hierarchical clustering has almost zero F-score at the end of ER phase when the datasets contain singleton entities and therefore suffer from poor progressiveness. Instead, we consider `Hybrid` as an adaptation of the state-of-the-art entity resolution strategy [24] that treats each internal node as a candidate cluster for its benefit calculation. In addition to these baselines, we plot the ideal curve that is assumed to know the ground truth but is required to generate the hierarchy based on evidence from triplet or equality oracle queries. It is allowed to leverage the ground-truth to optimize for progressiveness. This strategy is used as an empirical lower-bound to construct the hierarchy. We consider a variant (denoted by IdealTr) that denotes the ideal strategy for type-hierarchy generation and is allowed to use triplet queries only. To verify insertion of a new branch at any internal node, `ideal` requires $O(\alpha)$ queries, where α is the degree of the internal node. Since some datasets have a very high α, we use a threshold $\theta = \log n$ to insert the branch.

Variations. We use `HierER` to denote our Hierarchical algorithm that uses both pairwise and triplet queries for hierarchy construction. Additionaly, we consider the following variations. (b) `HierTr.Eq` is a sequential pipeline that uses triplet queries to generate a type-hierarchy (`Hier-Type`) and then uses equality queries to identify entity nodes. `HierTr` is the hierarchical algorithm that uses only the triplet queries for type-hierarchy construction using `Hier-Type` and does not identify entity nodes. (c) `HierER-Nb` is the same as `HierER` but does not perform benefit calculation for querying. It processes records in a randomized order.

Evaluation Metrics. We compare the performance of techniques by measuring progressive F-score. We consider all the identified pairwise relationships in the constructed hierarchy and map those to the relationships of the ground truth hierarchy to evaluate their contribution. Note that enumerating all $\binom{n}{2}$ pairs is not feasible for million scale datasets. However, we observe that the contribution of a record u to the pairwise relationships is the same as that of v whenever u.`parent` = v.`parent`. Using this property, we identify the super nodes of records that refer to the same leaf level cluster and then consider pairwise relationships between these super nodes to calculate the overall F-score. This optimization is highly efficient as the internal nodes are generally much fewer than n.

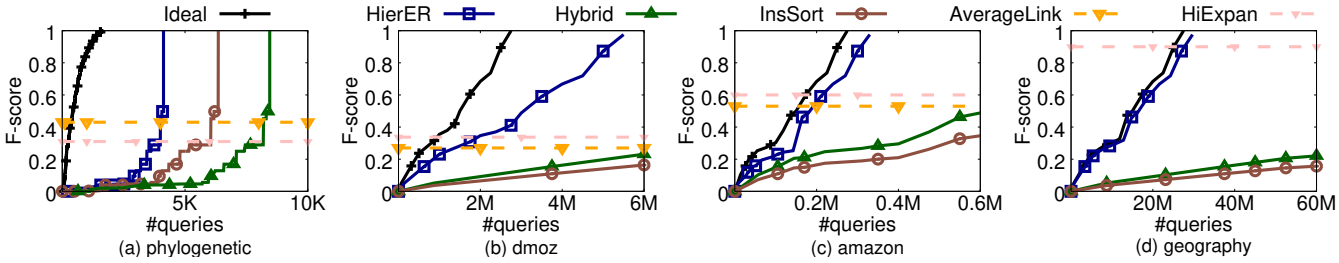


Figure 10: F-score vs #queries comparison for datasets where all records refer to distinct entities. In these datasets, `HierER` performs similar to its pipelined variants `HierTr.Eq` and `HierTr` since equality oracle does not provide any information.

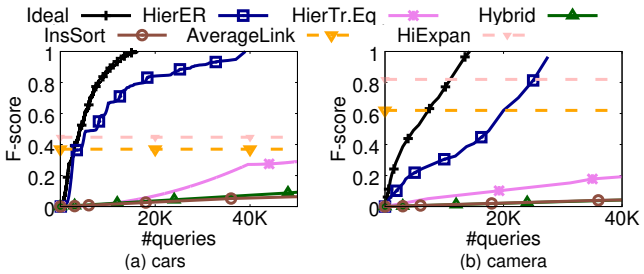


Figure 11: F-score vs #queries comparison for datasets with high-degree entity nodes.

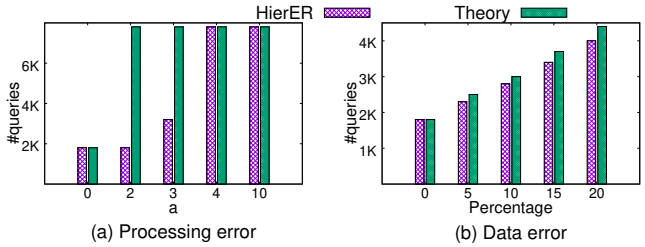


Figure 12: Query complexity for varying similarity noise.

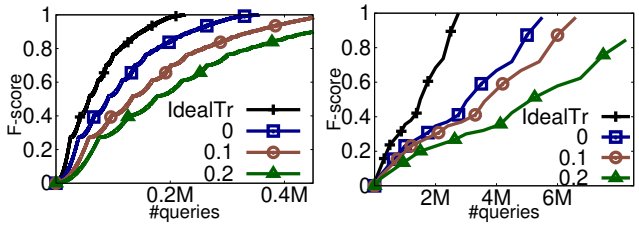
9.2 Result Quality

Figures 10 and 11 compare the F-score of `HierER` with other baselines on multiple datasets. Across all datasets, `HierER` achieves the highest progressive F-score and is closest to the `ideal` curve. `InsSort` and `Hybrid` achieve poor progressive F-score. These techniques require more than 5× the queries required by `HierER` to achieve comparable F-score across all large scale datasets. `AverageLink` generates a hierarchical clustering over the records without any oracle queries. This hierarchy achieves non-zero F-score but the mistakes in the constructed hierarchy are not corrected by additional queries. Additionally, it does not run on million scale datasets like `Geography`. `HiExpan` performs better than `AverageLink` for most datasets but does not achieve high F-score. It is sensitive to the initial structure provided as input and does not generalize if it does not contain all levels of the hierarchy. Due to the presence of noise in datasets, such automated techniques do not achieve high F-score.

We observe different behaviors across datasets. In `phylogenetic` dataset, the ideal requires $< 2n$ queries as majority of the internal nodes have two children and require less than 2 queries to be inserted. For each leaf level record, `InsSort` requires $O(\log n)$ queries to identify its location in the processed hierarchy. Therefore, `InsSort` requires $O(n \log n)$ queries and has poor progressive recall. `HierER` performs better than `InsSort` but requires much more queries than the `ideal` strategy. We examined the pairwise similarity values of records and observed that more than 85% of the records suffer from data error in similarity computation. Due to high noise in similarity values, `HierER` could not identify the existence of many internal nodes in the initial stages of resolution. This delay in identifying all internal nodes of the hierarchy affects the progressiveness. However, the overall complexity of `HierER` is around $O(n \log n)$. The nodes that do not suffer from error require fewer than two queries to get inserted but the nodes with data error require $O(\log n)$ queries. `Hybrid` requires more queries than `InsSort` due to the noise in similarity values. For `Hybrid`, the nodes that do not suffer from any noise require fewer than five queries to get inserted but some erroneous nodes require $O(t)$ queries, where t is the number of internal nodes in the hierarchy.

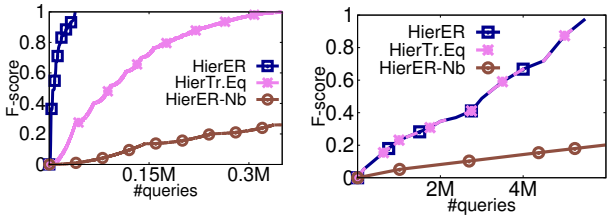
Among other datasets, most internal nodes have degrees much higher than 2. `HierER` achieves near-optimal progressive F-score and performs much better than all other baselines. Poor performance of `Hybrid` and `InsSort` is due to the high degree of internal nodes. For a given internal node (of degree α), `InsSort` queries the branches randomly, which requires $O(\alpha)$ queries to identify the correct branch. In contrast, `Hybrid` is beneficial for less noisy records but due to high degree of internal nodes, queries required to identify a new branch dominate the overall complexity.

Comparison with theoretical results. Among different datasets, we observe that the gap between `ideal` and `HierER` is highest for `Phylogenetic` dataset due to presence of noise in similarity values. To validate the effect of noise, Figure 12 considers synthetic similarities and compares the query complexity with the theoretically proven bounds in Section 7. Figure 12(a) simulates processing error, where each pairwise similarity $(s(u, v))$ is sampled independently according to a normal distribution with mean $\mu_{(u,v)}$ and variance σ^2. As discussed in Section 7, μ denotes the difference in expected similarity for pairs connected at different depths. To simulate different levels of noise, we considered $\sigma^2 = a\mu^2$ for varying values of a. The query complexity of `HierER` is the same as that of `ideal` (roughly $2n$) for $a < 3$. For higher noise, the query complexity increases but it plateaus at $O(n \log n)$. Figure 12(b) shows the query complexity vs data error tradeoff in `phylogenetic` dataset. In this experiment, a random sample of the nodes are simulated to be erroneous such that all pairwise similarities containing these records



(a) cars (b) dmoz

Figure 13: Varying oracle error



(a) cars (b) dmoz

Figure 14: Comparison between variants of HierER

are erroneous. In this case, we observe that the query complexity of the technique is directly proportional to the error.

Variants of the querying strategy. Figure 14 compares the progressive quality of HierER with a sequential strategy HierTr.Eq and a variant that does not use benefit calculation (HierER-Nb). HierER is better than the pipelined strategy (HierTr.Eq) for the cars dataset, where many records refer to the same entity. Such clusters of records can be quickly identified using equality oracle queries. However, HierER and HierTr.Eq have comparable progressive quality for dmoz due to absence of any pair of records that refer to the same entity. The no-benefit variant queries records in a randomized order and has the worst progressive quality as compared to the HierER and HierTr.Eq.

Oracle error. The experiment in Figure 14 assumes that the oracle answers all queries correctly. However, some oracle queries can be more difficult, leading to errors. In this experiment, we considered independent triplet error where each triplet is erroneous with probability p. To develop *robust* HierTr, we leverage the random graph toolkit [26] for each oracle query. Figure 13 shows the impact of noise in oracle answers on the queries required to construct the hierarchy. It shows that HierTr identifies the hierarchy correctly whenever error is less than 0.3 and the query complexity increases by $O(\log n)$, due to the redundancy introduced by [26].

Running Time. Table 2 compares the running time of HierER to reach 0.90 F-score as compared to other baselines. HierER has lower running time than other techniques for most large scale datasets due to the linear dependence of running time on the number of queries required. HierER requires less than 2 minutes on phylogenetic dataset and finishes in less than 12 hrs for geography dataset.

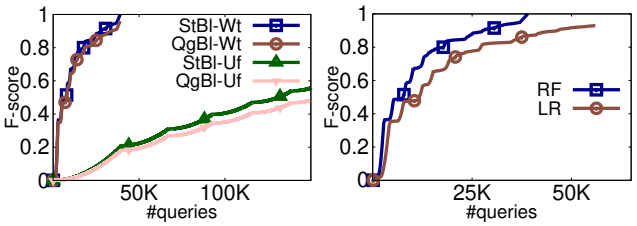
9.3 Ablation Analysis

In this section, we test the effectiveness of the default settings by varying blocking methods (Figure 15(a)) and classification techniques (Figure 15(b)).

Blocking. The default setting of HierER uses standard blocking for building blocks followed by TF-IDF based weighting mechanism

Table 2: Running Time Comparison

Dataset	HierER	InsSort	Hybrid	AverageLink
Phylogenetic	**1min 40sec**	1min 50sec	1min 53sec	2min 10sec
DMOZ	**1hr 23min**	4hr 47min	6hr 20min	3hr 17min
Cars	**45min**	3hr 25min	5hr 30min	3hr 5min
Camera	**52min**	2hr 47min	3hr 35min	3hr 10min
Amazon	**1hr 5min**	3hr 27min	3hr 55min	3hr 40min
Geography	**11hr 35min**	30hr 35min	34hr 35min	Did Not finish



(a) Blocking (b) Classifier

Figure 15: Ablation analysis with varying blocking and classification methods on cars dataset.

for meta-blocking (denoted by StBl-Wt in Figure 15(a)). We tested it with three variants (a) Q-gram based blocking [53] with TF-IDF weights (QgBl-Wt) (b) Q-gram based blocking with uniform weights (QgBl-Uf), and (c) standard blocking with uniform weights (StBl-Uf). Figure 15(a) shows that changing block building strategy from StBl to QGBl does not impact solution quality because QgBl is expected to perform better than StBl in datasets with spelling mistakes and cars dataset does not contain such mistakes. However, changing the block weights from TF-IDF to Uniform weights worsens the overall performance. These observatios are consistent with prior studies on blocking [27, 53] which show the benefits of TF-IDF weighting of blocks.

Classification. The default setting of HierER uses a random forest classifier, which has 0.97 F-score on cars dataset. Figure 15(b) compares the progressiveness of using random forest (RF) and logistic regression (LR) employed with HierER's querying strategy. The logistic regression classifier (F-score=0.82) is less accurate than Random Forest classifier on this dataset. Due to higher error rate in the oracle response of LR, it requires more queries to correct mistakes (as discussed in the oracle error paragraph above). Additionally, HierER's error correction mechanism is not able to correct all the mistakes made by LR and converges at a lower final F-score than random forest.

10 CONCLUSION

In this paper, we formalize the Hierarchical Entity Resolution problem using an Oracle. We propose HierER, an efficient query ordering strategy that leverages pairwise record similarities to prioritize triplet and equality oracle queries. We prove that HierER constructs the ground truth hierarchy in $O(n \log n)$ queries under reasonable assumptions of processing and data error models. We empirically demonstrate its effectiveness over various real world datasets.

ACKNOWLEDGEMENTS

This work was supported by NSF grants #2127309, 1652303, 1909046, and HDR TRIPODS 1934846 grants, a SEED PNR FLOWER grant 2021 and an Alfred P. Sloan Fellowship

REFERENCES

[1] Dmoz https://dmoz-odp.org/.
[2] Google vision api https://cloud.google.com/vision.
[3] Scikit-learn https://scikit-learn.org/stable/modules/clustering.html.
[4] World cities database https://simplemaps.com/data/world-cities.
[5] Raghavendra Addanki, Sainyam Galhotra, and Barna Saha. How to design robust algorithms using noisy comparison oracle. *arXiv preprint arXiv:2105.05782*, 2021.
[6] Alfred V. Aho, Yehoshua Sagiv, Thomas G. Szymanski, and Jeffrey D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.
[7] Mohit Bansal, David Burkett, Gerard De Melo, and Dan Klein. Structured learning for taxonomy induction with belief propagation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1041–1051, 2014.
[8] Jonathan Bragg and Daniel S Weld. Optimal testing for crowd workers. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 966–974, 2016.
[9] Gerth Stølting Brodal, Rolf Fagerberg, Christian NS Pedersen, and Anna Östlin. The complexity of constructing evolutionary trees using experiments. In *International Colloquium on Automata, Languages, and Programming*, pages 140–151. Springer, 2001.
[10] Daniel G Brown and Jakub Truszkowski. Fast error-tolerant quartet phylogeny algorithms. In *Annual Symposium on Combinatorial Pattern Matching*, pages 147–161. Springer, 2011.
[11] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, Jun 2020.
[12] Vaggos Chatziafratis, Rad Niazadeh, and Moses Charikar. Hierarchical clustering with structural constraints. *arXiv preprint arXiv:1805.09476*, 2018.
[13] Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Crowdsourcing for top-k query processing over uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):41–53, 2015.
[14] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 378–397. SIAM, 2018.
[15] Valter Crescenzi, Andrea De Angelis, Donatella Firmani, Maurizio Mazzei, Paolo Merialdo, Federico Piai, and Divesh Srivastava. Alaska: A flexible benchmark for data integration tasks, 2021.
[16] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 118–127, 2016.
[17] Susan Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Top-k and clustering with noisy comparisons. *ACM Trans. Database Syst.*, 39(4), December 2015.
[18] Xin Luna Dong and Divesh Srivastava. Big data integration. Morgan & Claypool, 2015.
[19] Eyal Dushkin and Tova Milo. Top-k sorting under partial order information. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 1007–1019, 2018.
[20] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.
[21] Ehsan Emamjomeh-Zadeh and David Kempe. Adaptive hierarchical clustering using ordinal queries. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 415–429. SIAM, 2018.
[22] Brian Eriksson, Gautam Dasarathy, Aarti Singh, and Rob Nowak. Active clustering: Robust and efficient hierarchical clustering using adaptively selected similarities. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 260–268, 2011.
[23] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
[24] Donatella Firmani, Barna Saha, and Divesh Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.
[25] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Hierarchical entity resolution using an oracle. In *https://hierarchicaler.github.io*.
[26] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Robust entity resolution using random graphs. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 3–18, 2018.
[27] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Efficient and effective er with progressive blocking. *The VLDB Journal*, pages 1–21, 2021.
[28] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.
[29] Debarghya Ghoshdastidar, Michaël Perrot, and Ulrike von Luxburg. Foundations of comparison-based hierarchical clustering. In *Advances in Neural Information Processing Systems*, pages 7454–7464, 2019.
[30] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 601–612, 2014.
[31] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
[32] Ilan Gronau, Shlomo Moran, and Sagi Snir. Fast and reliable reconstruction of phylogenetic trees with indistinguishable edges. *Random Structures & Algorithms*, 40(3):350–384, 2012.
[33] Anja Gruenheid, Besmira Nushi, Tim Kraska, Wolfgang Gatterbauer, and Donald Kossmann. Fault-tolerant entity resolution with the crowd, 2015.
[34] Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. So who won? dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 385–396, 2012.
[35] Amit Gupta, Rémi Lebret, Hamza Harkous, and Karl Aberer. Taxonomy induction using hypernym subsequences. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1329–1338, 2017.
[36] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.
[37] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Coling 1992 volume 2: The 15th international conference on computational linguistics*, 1992.
[38] Max Hopkins, Daniel Kane, Shachar Lovett, and Gaurav Mahajan. Noise-tolerant, reliable active classification with comparison queries. *arXiv preprint arXiv:2001.05497*, 2020.
[39] Jin Huang, Zhaochun Ren, Wayne Xin Zhao, Gaole He, Ji-Rong Wen, and Daxiang Dong. Taxonomy-aware multi-hop reasoning networks for sequential recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 573–581, 2019.
[40] Christina Ilvento. Metric learning for individual fairness. *arXiv preprint arXiv:1906.00250*, 2019.
[41] Sampath K Kannan, Eugene L Lawler, and Tandy J Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21(1):26–50, 1996.
[42] Ehsan Kazemi, Lin Chen, Sanjoy Dasgupta, and Amin Karbasi. Comparison based learning from weak oracles. *arXiv preprint arXiv:1802.06942*, 2018.
[43] Rolf Klein, Rainer Penninger, Christian Sohler, and David P Woodruff. Tolerant algorithms. In *European Symposium on Algorithms*, pages 736–747. Springer, 2011.
[44] Ngai Meng Kou, Yan Li, Hao Wang, Leong Hou U, and Zhiguo Gong. Crowdsourced top-k queries by confidence-aware pairwise judgments. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, pages 1415–1430, 2017.
[45] Zornitsa Kozareva and Eduard Hovy. A semi-supervised method to learn and construct taxonomies using the web. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1110–1118, 2010.
[46] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*, 2020.
[47] Yuning Mao, Jingjing Tian, Jiawei Han, and Xiang Ren. Hierarchical text classification with reinforced label assignment. *arXiv preprint arXiv:1908.10419*, 2019.
[48] Yuning Mao, Tong Zhao, Andrey Kan, Chenwei Zhang, Xin Luna Dong, Christos Faloutsos, and Jiawei Han. Octet: Online catalog taxonomy enrichment with self-supervision. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2247–2257, 2020.
[49] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 19–34, 2018.
[50] Roberto Navigli, Paola Velardi, and Stefano Faralli. A graph-based algorithm for inducing lexical taxonomies from scratch.
[51] James B Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109–129, 1997.
[52] Alexander Panchenko, Stefano Faralli, Eugen Ruppert, Steffen Remus, Hubert Naets, Cédrick Fairon, Simone Paolo Ponzetto, and Chris Biemann. Taxi at semeval-2016 task 13: a taxonomy induction method based on lexico-syntactic patterns, substrings and focused crawling. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1320–1327, 2016.
[53] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9):684–695, 2016.
[54] Vassilis Polychronopoulos, Luca De Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. Human-powered top-k lists. In *WebDB*, pages 25–30, 2013.
[55] Cristina Roquet, Sébastien Lavergne, and Wilfried Thuiller. One tree to link them all: a phylogenetic dataset for the european tetrapoda. *PLoS currents*, 6.

[56] Jiaming Shen, Zeqiu Wu, Dongming Lei, Chao Zhang, Xiang Ren, Michelle T Vanni, Brian M Sadler, and Jiawei Han. Hiexpan: Task-guided taxonomy construction by hierarchical tree expansion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2180–2189, 2018.

[57] Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. Adaptively learning the crowd kernel. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 673–680, 2011.

[58] Jakub Truszkowski, Yanqi Hao, and Daniel G Brown. Towards a practical o (n logn) phylogeny algorithm. *Algorithms for molecular biology*, 7(1):32, 2012.

[59] Antti Ukkonen. Crowdsourced correlation clustering with relative distance comparisons. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1117–1122. IEEE, 2017.

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

[61] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, pages 989–998, 2012.

[62] Victor Verdugo. Skyline computation with noisy comparisons. In *Combinatorial Algorithms: 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8–10, 2020, Proceedings*, page 289. Springer.

[63] Vasilis Verroios and Hector Garcia-Molina. Entity resolution with crowd errors. In *2015 IEEE 31st International Conference on Data Engineering*, pages 219–230. IEEE, 2015.

[64] Vasilis Verroios, Hector Garcia-Molina, and Yannis Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, pages 1133–1148, 2017.

[65] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[66] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *arXiv preprint arXiv:1208.1927*, 2012.

[67] Jingjing Wang, Changsung Kang, Yi Chang, and Jiawei Han. A hierarchical dirichlet model for taxonomy expansion for search engines. In *Proceedings of the 23rd international conference on World wide web*, pages 961–970, 2014.

[68] Peter Welinder, Steve Branson, Pietro Perona, and Serge Belongie. The multidimensional wisdom of crowds. *Advances in neural information processing systems*, 23:2424–2432, 2010.

[69] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[70] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 481–492, 2012.

[71] Jingru Yang, Ju Fan, Zhewei Wei, Guoliang Li, Tongyu Liu, and Xiaoyong Du. Cost-effective data annotation using game-based crowdsourcing. *PVLDB*, 12(1):57–70, 2018.

[72] Chao Zhang, Fangbo Tao, Xiusi Chen, Jiaming Shen, Meng Jiang, Brian Sadler, Michelle Vanni, and Jiawei Han. Taxogen: Unsupervised topic taxonomy construction by adaptive term embedding and clustering. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2701–2709, 2018.

[73] Dongxiang Zhang, Yuyang Nie, Sai Wu, Yanyan Shen, and Kian-Lee Tan. Multi-context attention for entity matching. In *Proceedings of The Web Conference 2020*, pages 2634–2640, 2020.

[74] Chen Zhao and Yeye He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*, pages 2413–2424, 2019.