# Hierarchical Entity Resolution using an Oracle

Sainyam Galhotra
UMass Amherst
sainyam@cs.umass.edu

Donatella Firmani
Roma Tre University
donatella.firmani@uniroma3.it

Barna Saha
UC Berkeley
barnas@berkeley.edu

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

## ABSTRACT

In many applications, entity references (i.e., records) and entities need to be organized in a hierarchy capturing type-subtype relationships. However, entity identification tasks such as Entity Resolution (ER) focus on the problem of recognizing which records refer to the same entity (e.g., the same product sold by different sellers) and ignore hierarchical type-subtype information between entities. In this paper, we design and study a *hierarchical* ER task, where records are clustered in a tree-like structure: at the lowest level of the hierarchy we solve an ER problem, but we also augment it with the hierarchical categorization of the resulting entities. We leverage triplet oracle queries that take three records as input and output the most similar pair(s), and develop `Hero`, a querying strategy that uses prior record pair similarities to minimize the number of oracle queries while maximizing the identified hierarchical structure. `Hero` can also leverage ER techniques to identify records that refer to the same entity and thus improve the reconstruction of the lowest level of the hierarchy. We show theoretically and empirically that `Hero` is robust under different similarity noise models and demonstrate empirically that `Hero` can scale up to million-scale datasets.

**PVLDB Reference Format:**
Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Hierarchical Entity Resolution using an Oracle. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

## 1 INTRODUCTION

In many applications, records (which refer to real-world entities) need to be organized in ways that capture entities and type relationships between entities. For example, a retail website that recognizes which records refer to the same product and organizes such products in the form of a product taxonomy can enable better search and recommendation. Consider a sample of products in Figure 1 collected from a retail website, and focus on records 6, 7, 8 and 9. A partial solution would be to identify that records 7 and 8 refer to the same entity – "Luxrite T8" tube light, while 6 and 9 refer to other products. A better solution would be to recognize that 9 - even if it refers to a different "F8T5/CW" tube light - is closer to 7 and 8 than to 6, which refers to a halogen bulb which is of a different entity type. Even more, when considering other records such as 10, which refers to a laptop, we can identify that 6 is now closer to 7, 8 and 9 (which are all "lighting" items) than to 10 and so on, hierarchically. Figure 2(a) shows the entire set of relationships

| Record id | Title | Price | Picture | Type |
|---|---|---|---|---|
| 1 | General Lighting, Soft White, 60W Equivalent A29 LED Light Bulb | 21.56 | | General LED |
| 2 | Sylvania Ultra LED Night Chaser, 250W Equivalent, Replacement for Halogen Flood SpotLight | 20.64 | | General LED |
| 3 | Kasa Smart Light Bulb, LED Smart WiFi Alexa Bulbs | 13.99 | | Smart LED |
| 4 | Philips Hue White 4-Pack A19 LED Smart Bulb, Bluetooth Zigbee compatible | 49.99 | | Smart LED |
| 5 | GE Lighting 38-watt Halogen Floodlight Bulb with Medium Base | 31.99 | | Halogen light |
| 6 | Lithonia Lighting Mini Single-Head Flood Light 150-Watt Double Ended Quartz Halogen Lamp | 13.99 | | Halogen light |
| 7 | Luxrite T8 Fluorescent Tube Light Bulb, Cool White | 29.99 | | Tube light |
| 8 | Luxrite U Bend LED Tube Light, T8 T12, 18W (32WEquivalent), 2100 Lumens, Direct or Ballast Bypass | 24.99 |  | Tube light |
| 9 | F8T5/CW 8W T5 12" Cool White 4100k FluorescentLight Bulb | 9.99 |  | Tube light |
| 10 | Sony VAIO Core i7 17.3-Inch Laptop | 395 | | Laptop |
| 11 | Sony VAIO EJ2 Series VPCEJ28FX/B 17.3-InchLaptop | 435 | | Laptop |
| 12 | Sony vaio i7 laptop, 17 inch | 399 | | Laptop |

**Figure 1: Example collection of products. Records** 1, 2 **refer to Regular LED bulbs,** 3, 4 **to Smart LED bulbs,** 5, 6 **to halogen bulbs,** 7, 8, 9 **to tube lights and** 10, 11, 12 **to laptops.**

for the records in Figure 1. In this context, types can be thought of as clusters of entities, possibly included in other more general types, and entities can be thought of *sui generis* types consisting only of themselves. Reconstructing the entire set of hierarchical relationships can be challenging because types and entities may not be known a priori (i.e., some may need to be discovered during the process) and even the number of types or their size distribution may not be known; some types might have many entities in the data collection while others may be *niche* types with few entities.

**Related problems.** 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 building block of data integration pipelines [17]. ER focuses on the problem of identifying which records refer to the same entity, and ignores type information. 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) [11, 19, 35] and taxonomies [46, 59, 61]. In such applications, records refer to different entities, and thus clustering methods ignore entity resolution. Moreover, they build almost binary hierarchies, where every node has two (or slightly more than two)

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

(a) Ground truth hierarchy (b) Example triplet query (c) Hierarchy with triplet queries

**Figure 2: (a) Hierarchical relationships for products in Figure 1. Dark gray nodes like $e_9$ represent entities. E.g., records 10, 11 and 12 indeed refer to the same 'Vaio laptop' (denoted by $e_9$), sold by different vendors. Light gray nodes ($u_1 - u_7$) represent types. Deepest light gray nodes ($u_1, u_2, u_4, u_5$) which are parents of dark gray nodes, correspond directly to the "Type" column of Figure 1. E.g., $u_2$ corresponds to Smart LED. (b) Example of a triplet query about nodes 1, 4 and 11. The correct answer is "A". (c) Hierarchy constructed with triple queries does not identify the entity labelled internal nodes.**

children. Thus, neither ER nor Hierarchical Clustering techniques can by itself solve our problem completely.

**Hierarchical ER.** In this paper, we investigate a new *hierarchical ER* task, where records need to be clustered in a tree-like structure capturing entities as parents of leaf records and types from the level immediately above. In order to deal with high heterogeneity of records, we assume access to an *oracle* that provides an elegant abstraction of a domain expert or an accurate classifier trained on the dataset at hand. We use queries of two forms:

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

The above query forms have been used separately for solving ER [22, 23, 55] and hierarchical clustering [19]. In all these studies, how many queries are asked is a major consideration. We set the goal of our problem to maximize the *progressive F-score* of the output hierarchy: if we imagine F-score as a curve with respect to the sequence of oracle queries, the progressive F-score can be thought of the area under this curve. Maximizing progressive F-score ensures that at any point of time we can return the best hierarchy with respect to the resources spent. We extend the traditional definition of F-score to our hierarchical ER setting and measure it after every oracle query (irrespective of the form).

**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 [19] 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 an ER technique like [22] first and post-processing entities after that to detect types can be very efficient in case of large entities but can require $O(n^2)$ queries to identify small entity clusters, before even starting to process types.

**Our contributions.** We develop two algorithms for **H**ierarchical **E**ntity **R**esolution using an **O**racle, called `Hero-Type` and `Hero`,

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

Both `Hero-Type` and `Hero` 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 display 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 `Hero-Type` and `Hero` 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, `Hero-Type` and `Hero` require $O(n \log n)$ oracle queries and achieve optimal progressive F-score with high probability. Experimental results show that `Hero-Type` and `Hero` can scale up to datasets with millions of records and outperform baseline solutions both in efficiency and effectiveness. 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 in [23].

**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 formalize the notation and define the problem statement. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a collection of $n$ records. Let $H^*$ denote the ground truth of our hierarchical ER problem i.e., a rooted tree containing the actual type-subtype and co-reference (i.e., referring to the same entity, or equality) relationships and $n$ leaves, each corresponding to a distinct record $v_i \in V$. We distinguish

internal nodes of $H^*$ i.e., all its nodes except the leaves, into *type nodes* $u \in T$ and *entity nodes* $e \in E$. An entity can be thought of as a set of records that co-refer to the entity, while a type 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 $H^*$ more formally.

*Definition 2.1 (Laminar Family).* A family of sets $C^*$ is laminar iff $\forall X_1, X_2 \in C^*$, 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 $C^*$ such that each set in $C^*$ 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, $\langle \texttt{r} : \{v\} \rangle \in C^*, \forall v \in V$ and $\langle \texttt{t} : V \rangle \in C^*$. 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 $C^*$ where an internal node of the hierarchy (say $u$) is equivalent to the set $C \in C^*$ containing all the leaf-level descendants of $u$ and vice versa. In this formulation, we assume that all type nodes of the hierarchy have at least two children. A type node that has a single child is redundant and can be ignored. However, entity nodes can have only one child if it has a single entity reference (record).

*Example 2.2.* The hierarchy in Figure 2(a) is equivalent to a laminar family of 28 labelled sets where 12 sets are labelled `record`, 9 are labelled `entity` (corresponding to $e_1$-$e_9$) and 7 are labelled `type` ($u_1$-$u_7$). Some of these sets are listed below.

$$\{\langle r : \{1\}\rangle \cup \langle r : \{2\}\rangle \cup \langle e : \{1\}\rangle \cup \langle e : \{2\}\rangle \cup \langle t : \{1, 2\}\rangle \cup \langle t : \{1, 2, 3, 4\}\rangle\}$$

The Lowest Common Ancestor (`lca`) of records $v_1, v_2$ is defined as the common ancestor of both $v_1$ and $v_2$ that is farthest from the root. If the `lca` is an entity, 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.

The 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 nodes have equal depth and the `lca` of the third pair has depth not less than the depth of the other two `lca`'s. Without loss of generality,

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

In case all three `lca`'s have same depth, then $v_1$, $v_2$ and $v_3$ are guaranteed to belong to three different descendant-branches of the node corresponding to $\texttt{lca}(v_1, v_2) = \texttt{lca}(v_1, v_3) = \texttt{lca}(v_2, v_3)$.

*Example 2.4.* Consider the records 1, 4, 11 in Figure 2(a). We observe that $\texttt{lca}(1, 4) = u_3$ is present deeper in hierarchy as compared to $\texttt{lca}(1, 11) = u_7$, i.e., $\texttt{depth}(\texttt{lca}(1, 4)) > \texttt{depth}(\texttt{lca}(1, 11))$ and $\texttt{lca}(1, 11) = \texttt{lca}(4, 11)$. For the records (1, 5, 7), we observe that all pairwise `lca`'s have the same depth and thus the three records are present in different descendant branches of $\texttt{lca}(1, 5) = u_6$.

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 $\texttt{depth}(\texttt{lca}(v_1, v_2)) = \texttt{depth}(\texttt{lca}(v_1, v_3)) < \texttt{depth}(\texttt{lca}(v_2, v_3))$ and $q_t(v_1, v_2, v_3) = \phi$ if $\texttt{lca}(v_1, v_2) = \texttt{lca}(v_1, v_3) = \texttt{lca}(v_2, v_3)$.

*Example 2.6.* Consider again the records 1, 4, 11 in Figure 2(a). Possible answers to the query "which records among 1, 4 and 11 are most similar?" are shown in Figure 2(b). The arrangement A corresponds to answer "1 and 4 are closer" or equivalently "11 is the farthest", while other arrangements B and C identify 1 and 4 as the farthest record respectively. Finally, the arrangement D corresponds to $q_t(1, 4, 11) = \phi$ i.e., all three nodes are equally close.

Note that 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, our best result can be as in Figure 2(c), that is, without entity nodes $e_1, \ldots, e_9$ and with misinterpretation of $e_7$ and $e_9$ as type nodes. In order to fill in the missing information, we consider a black box that outputs whether any pair of records refers to the same entity.

*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.

A recent work [19] showed that any strategy that queries all $\binom{n}{3}$ triples can recover the underlying type-subtype hierarchy uniquely assuming that the triplet oracle makes no mistakes.[1] Since the maximum number of equality queries possible are $O(n^2)$, the $\Theta(n^3)$ upper-bound in [19] also holds for hierarchical ER. Indeed, this bound entails that, after querying $\binom{n}{3}$ distinct triples, we do not need to discover new type nodes. Thus, we only need to add the entity nodes, for which additional $O(n)$ equality queries are sufficient, assuming that the equality oracle makes no mistakes.

**Progressive F-score.** A naive way to compare performance of different hierarchical ER strategies is to consider which of the total $\Theta(n^3)$ relationships (i.e., $\binom{n}{3}$ type-subtype plus $\binom{n}{2}$ equality relationships) have been correctly identified by each of the strategies and which have not. However, such an approach can be infeasible even for medium-sized datasets. Therefore, we introduce an extension of the popular ER metric *F-score* in our hierarchical setting: we first define the notion of `t-ancestor` relationship and then show how to use it to compute the F-score of the output hierarchy $H$.[2]

*Definition 2.8 (t-`ancestor` relationship).* A pair of records $(u, v)$ are considered to satisfy a `t-ancestor` relationship if their `lca` is at most t edges away from both $u$ and $v$. `1-ancestor` relationship is equivalent to an equality (i.e., co-reference) relationship.

[1] [19] also showed that $O(n \log n)$ queries are sufficient for binary hierarchies.

[2] 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 3: Our hierarchical ER workflow. The blocks in red represent our novel contributions, whereas the blocks in blue are implemented with methods available in literature.**

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 mechanism 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 [22, 23].

**Problem statement.** Now, we define our problem statement.

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

Problem 1 assumes that the querying strategy has no knowledge about the ground truth hierarchy $H^*$. However, in many applications, additional information about the hierarchy is often available and can be used to save oracle queries. Notable variants of Problem 1 include the case where the degree of each internal node of $H^*$ is $\leq 2$ (*Binary Hierarchical ER*) and the case where the degree of each entity node of $H^*$ is 1, i.e., all entities are singletons (no duplicate records). In particular, the latter variant can be solved by asking only ordinal queries to find type nodes, as singleton entity nodes can be added as a post-processing step without extra queries. We provide more discussion for variants in the full version [4].

Finally, similarity values can be calculated using textual descriptions in records 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. We provide theoretical analysis of our method under reasonable assumptions for $s$ (i.e., 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 3. Modules are described below.

**Blocking and Similarity.** In order to reduce the number of pairs considered for similarity computation, we perform *blocking* [42]. Blocking is a widely used operation in ER literature that assigns records to different blocks, computes similarity only for pairs that share at least one block (typically much less than $\binom{n}{2}$) and provides a default value (e.g., $s(v_1, v_2) = 0$) for all the remaining pairs. Details about the blocking method and similarity functions used in our experiments are provided in Section 9.

**Probability Calibration.** Similarity values may not be directly interpretable as probability distributions over the possible oracle responses. We use the calibration approach in [22, 58] 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. In a nutshell, we *bucketize* queries according to similarity values and ask the oracle a few queries, by sampling from each bucket. Finally, we assign the same probability to all the queries in the same bucket, according to the frequencies of oracle responses [22, 58].

- For the equality oracle, buckets correspond to similarity ranges [58], e.g., $[0, 0.1), [0.1, 0.2) \ldots [0.9, 1.0]$. Consider for example a similarity bucket $[0, 0.1)$ and suppose that 15% of sampled pairs of records from that bucket receive answer $T$ (i.e., they refer to the same entity). Thus, for every $v_1, v_2 \in V : s(v_1, v_2) \in [0, 0.1)$ we set $Pr(q_e(v_1, v_2) = T) = 0.15$.
- For the triplet oracle, we extend [58] to construct two dimensional buckets corresponding to a pair of similarity ranges. The bucket for a triplet is determined by the maximum and minimum similarity of the record pairs in the triplet. For example, consider a triplet $(v_1, v_2, v_3)$ such that $s(v_1, v_2) = 0.95$, $s(v_1, v_3) = 0.35$ and $s(v_2, w_3) = 0.25$. The triplet $(v_1, v_2, v_3)$ is placed in the bucket $([0.9 - 1], [0.2, 0.3))$. The probability $Pr[q_t(v_1, v_2, v_3) = \phi]$ is defined as the fraction of triplets in the samples that were labelled $\phi$ by the oracle. Since $(v_1, v_2)$ has the highest similarity, we estimate probability $Pr[q_t(v_1, v_2, v_3) = v_3]$ as the fraction of times the highest similarity pair is labelled closest by the oracle. Similarly, $Pr[q_t(v_1, v_2, v_3) = v_1]$ is the fraction that the lowest similarity pair is labelled closest and $Pr[q_t(v_1, v_2, v_3) = v_2] = 1 - Pr[q_t(v_1, v_2, v_3) = v_1 \text{ or } v_3 \text{ or } \phi]$.

**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 [55, 56] that equality relationships can be easily inferred via transitive closure.[3] 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 `Hero-Type` and

[3] 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 $\alpha$
1: **if** $|X| == 1$ **then return** X
2: $\bar{H}_X \leftarrow$ `new_node()`
3: **if** $|X| \leq \log n$ **then**
4: $\quad C \leftarrow$ `construct_partition1`$(X, s, \alpha)$
5: **else**
6: $\quad C \leftarrow$ `construct_partition2`$(X, s, \alpha)$
7: **for** $C \in C$ **do**
8: $\quad \bar{H}_c \leftarrow$ `generate_similarity_hierarchy`$(C)$
9: $\quad \bar{H}_X.add_children(\bar{H}_C)$
10: $\bar{H}_X \leftarrow$ MergeInternal$(\bar{H}_X)$
11: **return** $\bar{H}_X$

`Hero` approaches in Section 6. Different implementations of this module are compared by their progressive F-scores in Section 9.

**`Hero` strategies.** Given the workflow in Figure 3, `Hero-Type` and `Hero` can be thought of as oracle strategies that leverage the initialization and inference methods introduced in this paper and implement the following principles to maximize progressive F-score.

- *Internal node discovery.* `Hero-Type` and `Hero` give high priority to queries that enable the discovery of new internal nodes. Indeed, identifying all the internal nodes between the root node and the leaf nodes corresponding to such records is important to provide optimal progressive behaviour.
- *Large entities.* `Hero-Type` and `Hero` 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 literature [22, 23] since asking queries in non-increasing order of entity sizes provides the maximum gain in progressive recall.
- *Connectivity.* `Hero-Type` and `Hero` give high priority to queries that can 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.

**Results overview.** In Section 7, we first analyze the effectiveness of initialization to recover the ground truth hierarchy, under two representative noise models for the similarity values, that we refer to as *processing error* and *data error*.

- Processing error captures erroneous similarity values for random pairs, independent of one another.
- Data error captures erroneous similarity values for random records, thus affecting all the pairs involving that record.

We show that under reasonable error (both processing and data error), `Hero` recovers the underlying hierarchy correctly in less than $O(n \log n)$ queries and achieves better progressive F-score as compared to baseline pipelines. We also demonstrate in Section 7 that `Hero` can achieve the best progressive F-score with high probability, and provide detailed experimental results in Section 9.

## 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 $s : V \times V \rightarrow [0, 1]$. We formally analyze the quality of this algorithm in Section 7.

**Figure 4: Example candidate hierarchy (with $\alpha = 2$) where the internal node corresponding $\{v_3, v_4, v_5, v_6\}$ is merged with its children ($\{v_3, v_4\}$ and $\{v_5, v_6\}$) by MergeInternal.**

**Algorithm 2** construct_partition1

**Require:** records $X$, similarity $s$, degree $\alpha$
1: **if** |X|==1 **then**
2: $\quad$ **return** X
3: $v_1 \leftarrow$ `choose_random`$(X)$
4: $C \leftarrow \{X\}, S \leftarrow \{v_1\}$
5: **while** $|S| < \alpha$ **do**
6: $\quad v_i \leftarrow$ `identify_minimum_similarity`$(X, C, s)$
7: $\quad S \leftarrow S \cup \{v_i\}$
8: $\quad C \leftarrow$ `Assign`$(X, S)$
9: **return** $C$

Algorithm 1 takes a collection of records $X \subseteq V$ as input along with pairwise similarities $s$ and an optional parameter $\alpha$. (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 $\alpha$ clusters (denoted by $C$) by `construct_partition1` if $|X| < \log n$ and `construct_partition2` otherwise (lines 3-6). The clusters $C$ capture the different children branches of the root $\bar{H}_X$. The candidate hierarchy on each partition $C \in C$ 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 $\alpha$. To identify such nodes, MergeInternal post-processes $\bar{H}_X$ to consolidate internal nodes that have higher likelihood of referring to same node. This procedure processes 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 the calibrated probability distribution of triples that share the two `lca`'s.

*Example 4.1.* Figure 4 shows an example recursion for $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ where the first iteration of the recursion has $X = V$ and $C = \{\{v_1, v_2\}, \{v_3, v_4, v_5, v_6\}\}$.

**`construct_partition1`.** Algorithm 2 presents a greedy algorithm to identify the different branches of the set $X$. This algorithm is similar to the greedy k-center algorithm [27] where each branch is initialized by a seed node and remaining nodes are assigned to the closest seed nodes. $S$ denotes the set of seed nodes and $C$ denotes a partitioning of $X$ denoting the records assigned to respective branches. Given a set of records $X$, it initializes the first branch by randomly choosing a leaf level record as a seed (line 3) and all records in $X$ are assigned to the first seed. The subsequent

**Algorithm 3** construct_partition2

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

seed nodes are identified by choosing the record that has minimum similarity from its assigned seed record (line 6). All records in $X$ are re-assigned to the branches where each record is assigned to the closest seed node (line 8). 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 `construct_partition2` subroutine that is proven to be accurate when the different branches contain more than $\log n$ records.

**`construct_partition2`.** Algorithm 3 presents the pseudo code 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 $\alpha$ 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 `AssignCount`, 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

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

**Figure 5: Collection of hierarchies for two different sets of queries, where the dashed lines denote equivalence edges and solid lines capture ancestor-descendant relationship.**

**Algorithm 4** `identify_inferred`

**Require:** $\mathcal{H}$, Query $q \equiv q_t(x, y, z)$
1: $H \leftarrow \texttt{initialize_hierarchy}(x, y, z, q_t(x, y, z))$
2: $\mathcal{H} \leftarrow \mathcal{H} \cup \{H\}$
3: $L \leftarrow \{H\}, \texttt{newlyAdded}(H) = \{x, y, z\}$
4: **while** $H \in L$ **do**
5:  $L', \mathcal{H} \leftarrow \texttt{merge}(H, \mathcal{H})$
6:  $L \leftarrow L \cup L'$
7:  Remove $H$ if $H$ is a sub-tree of some $H' \in \mathcal{H}$
**return** $\mathcal{H}$

relationships that are either queried or can be inferred[4] from $Q$ can be answered from one of the constructed hierarchies. To better understand this behavior, we mathematically characterize the information available from a triplet query.

Consider a query $q_1 \equiv q_t(u, v, w)$ that returns $w$. To interpret $q_1$, we define three variables corresponding to the lca's of involved record pairs $(u, v), (u, w)$ and $(v, w)$. Using these variables, $q_1$ can be interpreted as a collection of the following inequalities.

```
depth(lca(u,v)) > depth(lca(u,w)) = depth(lca(v,w))
                  lca(u,w) = lca(v,w)
```

The first equation characterises a relation between the relative distance of each lca from the root node and the second equation is an equivalence relation between the two lca's. Each query can be written in the form of such inequality constraints over at most $\binom{n}{2}$ lca variables. Consider another query $q_2 \equiv q_t(u, w, x) = x$ which can be written as

```
depth(lca(u,w)) > depth(lca(u,x))=depth(lca(w,x))
                  lca(u,x) = lca(w,x)
```

Using the equations of $q_1$ and $q_2$, we can infer that

```
depth(lca(u,v)) > depth(lca(u,w)) > depth(lca(u,x))
```

With this evidence we can place the four records $u, v, w$ and $x$ on the same branch (Figure 5(a)). This example describes the merge operation over two hierarchies where the node $x$ is inserted into the first hierarchy based on response to query $q_2$. However, if the query $q_3 \equiv q(u, v, x)$ returns $x$ then the equations are as follows.

```
depth(lca(u,v)) > depth(lca(u,x))=depth(lca(v,x))
                  lca(u,x) = lca(v,x)
```

In this case we cannot infer if $\texttt{depth}(\texttt{lca}(u, w)) > \texttt{depth}(\texttt{lca}(u, x))$. Therefore, we cannot merge the hierarchies corresponding to $q_1$ and $q_3$ at the moment but with more queries like $(u, w, x)$, we will get enough evidence to merge the hierarchies following the same procedure. Following this notation, an equality query of the form

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

**Algorithm 5** `merge`

**Require:** $H, \mathcal{H}$
1: $E' \leftarrow$ `identify_new_equivalence(newlyAdded(`$H$`))`
2: $L \leftarrow \phi$
3: **for** $H', E \leftarrow E'$ **do**
4:  **if** $edges(H, H') \geq 2$ **then**
5:  $M \leftarrow \phi$
6:  **for** $u \in leaves(H) \setminus leaves(H')$ **do**
7:  **if** `insert(`u,$H'$`)` **then**
8:  $M \leftarrow M \cup \{u\}$
9:  **if** $M \neq \phi$ **then**
10:  $L \leftarrow L \cup \{(H'\}$
11:  `newlyAdded(`$H'$`)` $= M$
**return** $L, \mathcal{H}$

**Algorithm 6** `IdentifyBranch`

**Require:** Leaf level record $v$, Internal node $u$, threshold $t$
1: $C \leftarrow u.children()$, $j \leftarrow 0$
2: **for** $i = 0, \ldots, \lfloor (|C| - 1)/2 \rfloor$ **do**
3:  $c_1 \leftarrow C[2 * i]$, $c_2 \leftarrow C[(2 * i + 1)\% \, |C|]$
4:  $l_1 \leftarrow c_1$`.leaves[0]`
5:  $l_2 \leftarrow c_2$`.leaves[0]`
6:  $o \leftarrow q_t(v, l_1, l_2)$, $j \leftarrow j + 1$
7:  **if** $o == l_1$ **then return** $c_2, j$
8:  **if** $o == l_2$ **then return** $c_1, j$
9:  **if** $o == v$ **then return** $v, j$
10:  **if** $i \geq t$ **then**
11:  break
**return** Add a new Branch, $j$

**Algorithm 7** FindSibling

**Require:** Record $v$, Hierarchy $H$
1: $S \leftarrow$ `nodes(`$H$`)`
2: **while** $|S| > 1$ **do**
3:  Identify $u \in S$ such that $\max_{c \in children(u)} \{S \cap H_c, S \setminus leaves(u)\}$
4:  $o =$`IdentifyBranch(`$v$, $u$, $|v.children()|$`)`
5:  $S \leftarrow$ `update(`$S$, $o$`)`
**return** $S$

$q_4 \equiv q_e(u, v) = T$ reveals that $u$ and $v$ refer to same entity and is same as merging the nodes corresponding to $u$ and $v$ in $\mathcal{H}$. All such relationships can be inferred via transitive closure (see [55, 56] for more details). We now focus on inference from triplet queries and present an efficient approach to maintain the collection of hierarchies $\mathcal{H}$ over the queries $Q$ where a node may be present in multiple hierarchies. All equivalent copies of a node are assumed to be 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. This collection of hierarchies $\mathcal{H}$ is sequentially updated with the addition of each query to $Q$.

Algorithm 4 presents the pseudo code that considers the 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 $L$ of modified hierarchies is initialized with $H$ (lines 2-3). The list L contains all the hierarchies that have been modified due to the insertion of $q$. In addition to list $L$, we maintain a mapping of hierarchies to a set of records that have been newly added into the hierarchy (`newlyAdded`(H)) since it was last processed by Algorithm 4. The algorithm then iterates over $L$ trying to merge each of the modified hierarchies (Algorithm 5) with other hierarchies already present in $\mathcal{H}$ (lines 4-7). The merge operation returns a new list of modified hierarchies $L'$ which are considered for subsequent iterations of `merge` operation and the process continues until all hierarchies in $L$ have been processed.

The `merge` subroutine efficiently identifies the hierarchies $H' \in \mathcal{H}$ that can be merged with $H$[5]. This component first identifies new equivalence edges between the newly inserted nodes of $H$ (identified by `newlyAdded`(H)) and other hierarchies in $\mathcal{H}$. Let these edges be $E'$. It then iterates over the hierarchies $H' \in E'$ and tries to insert leaf level records from $H$ to $H'$ with the help of query responses in $Q$. This insertion of nodes operates in the same way as $x$ is added in Figure 5. If a record is successfully inserted, $H$ is added to $L'$ and the mapping of newly added records is also updated.

# 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 `Hero` algorithms that achieve optimal progressiveness.

[5]Note that all clusters are maintained with union-find data structure and each record maintains an adjacency list of parent-child relationships along with equivalence edges, helping in efficient discovery of equivalence edges across hierarchies.

## 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 $v$ along with 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 narrow down the search for the exact location to insert $v$ in $H$.

Algorithm 6 presents the pseudo code. It considers all the 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 $(l_1, l_2)$ have the property that $\texttt{lca}(l_1, l_2) = u$. If a query $(v, l_1, l_2)$ returns $l_1$ then $\texttt{depth}(\texttt{lca}(v, l_2)) > \texttt{depth}(\texttt{lca}(l_1, l_2))$, implying $v$ belongs to the same branch as $l_2$. Similarly, the branch is also identified if the oracle returns $l_2$ or $v$. However, if the query returns $\phi$, 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 $\phi$, $v$ is attached as a separate branch to $u$. $t$ is used an early stopping threshold to not explore all children branches. If $t < \lceil |C|/2 \rceil$ and the algorithm returns the addition of a new branch then $v$ is guaranteed to belong to the subtree rooted at $u$. We allow an extension of this method, such that if an additional edge $(u, u_1)$, where $u_1$ is a child of $u$ is input, then $u_1$ is considered as the first candidate in the list $C$ containing $u$`.children()` to check if $v$ is present in same branch as $u_1$. If $v$ is seen to be present in this branch then `IdentifyBranch`$(v, u_1, t)$ 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 FindSibling subroutine (Algorithm 7) takes a leaf-level record $v$ as input and identifies a sibling of $v$ in a hierarchy $H$. FindSibling queries $v$ with an internal node $u$ (using `IdentifyBranch`) 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 `IdentifyBranch`$(u, v)$

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 $\alpha$.

**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 either 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 $\texttt{IdentifyBranch}(v, u)$ with a difference that the similarity to probability mapping is 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. The internal location of $v$ according to $\bar{H}$ is considered to have maximum benefit. For equality queries, `benefit_equality` reuses the benefit calculation method in [22] to sort all the entity nodes in decreasing order of `benefit`. This method returns top-$\tau$ entity nodes that have high benefit (where $\tau = \log n$). We use the same parameters as [22]. We present the pseudocode of `benefit_triplet` and `benefit_equality` in the full version [4].

## 6.2 Hero algorithms

Algorithm 8 presents the pseudo code for `Hero-Type`. 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 $v$, $E(v) = \sum_u s(u, v)$ [22]. The algorithm then proceeds in two phases. The first phase (lines 2-5) iteratively inserts records into the processed hierarchy $H$ and achieves 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 `Hero-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 `IdentifyBranch` to attach a new branch. Such unverified nodes are re-processed in the second phase using FindSibling to find their exact location in the hierarchy. We now describe the insertion algorithm in detail.

**`Hero-TypeInsertion` phase.** Algorithm 9 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 8** `Hero-Type`

**Require:** records $V$, similarity $s$
1: $\bar{H} \leftarrow$ `generate_similarity_hierarchy`($V$)
2: $L \leftarrow$ `get_expected_cluster_size`($V$)
3: $P, H, P_t \leftarrow \phi$
4: **for** $v \in L$ **do**
5: $\quad H, P, P_t \leftarrow$ `Hero-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: $\quad H \leftarrow$ FindSibling($v$, $root$)
9: **return** $H$

**Algorithm 9** `Hero-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: $S \leftarrow initializeCandidates(H)$
4: **for** $b \in B \cap S$ **do**
5: $\quad o, t \leftarrow$ `IdentifyBranch`($v$, $b$, $\tau - i$)
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: $\quad\quad u \leftarrow$ FindSibling($v$, $S$, $H$)
18: $\quad\quad$ break
19: $H \leftarrow$ `ExpandBranch`($v$, $H$, $\bar{H}$)
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 questions involving $v$ exceeds $\gamma$, where $\gamma$ refers to the estimate of queries required by FindSibling to insert $v$ (lines 16-18). In this case, the benefit based querying is ignored and FindSibling subroutine is used to process $v$. (iii) the number of questions involving $v$ exceeds $\tau$ (early stopping threshold, set to $\log n$). If the number of questions involving $v$ exceeds $\tau$ then $v$ is attached to the internal node that belongs to $S$ and is present closest to the root node (lines 11-15). In this case, $v$ is marked tentative (line 13). Additionally, if the subtree rooted at $v.parent()$ contains only tentative nodes, then it is also labelled as tentative as its exact location has not been finalized.

Once the record has been inserted, the `ExpandBranch` 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 $\texttt{lca}(v, v') = u$ and inserts $v'$ into the hierarchy $H$.

**Extension to `Hero`.** `Hero` uses the equality oracle in addition to the triplet oracle to identify record pairs that refer to the same entity. `Hero` proceeds similar to `Hero-Type` (Algorithm 8) with a difference that it uses `HeroInsertion` (Algorithm 10) to insert the record $v$ instead of `Hero-TypeInsertion` in the first phase. It calculates the benefit of $v$ with respect to triplet queries (same as `Hero-Type`) and equality queries (same as `Hybrid` [22]). The sorted list of all candidate locations to insert $v$ are then sorted in non-increasing

**Algorithm 10** `HeroInsertion`

**Require:** record $v$, Hierarchy $H$, Processed records $P$, $P_t$, candidate hierarchy $\bar{H}$

1: $B_t \leftarrow$ `benefit_triplet`$(v, H, P)$
2: $B_e \leftarrow$ `benefit_equality`$(v, H, P)$
3: $B \leftarrow Sort(B_e \cup B_t)$
4: $S \leftarrow initializeCandidates(H)$, $i \leftarrow 1$
5: **for** $b \in B \cap S$ **do**
6:  **if** $b \in B_e$ **then**
7:  **if** $q_e(v, b)$ **then**
8:  Insert $v$ to $b$'s cluster and $P \leftarrow P \cup \{v\}$
9:  break
10:  **else**
11:  $o, t \leftarrow$ `IdentifyBranch`$(v, b, \tau - i)$
12:  $i \leftarrow i + t$, $S \leftarrow update(S, o)$
13:  **if** $|S| = 1$ **then**
14:  Insert $b$ as a sibling of $S$
15:  break
16:  **if** $i \geq \tau$ **then**
17:  Attach $v$ to the candidate with lowest depth
18:  $P_t \leftarrow P_t \cup \{v\}$
19:  **return** $H, P, P_t$
20:  break
21:  **if** $i \geq \gamma$ **then**
22:  $u \leftarrow$ FindSibling$(v, candidates, H)$
23:  break
24: $H \leftarrow$ `ExpandBranch`$(v, H, \bar{H})$
25: $P \leftarrow P \cup \{v\}$
26: **return** $H, P, P_t$

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

**Figure 6: Similarity distributions for pairs at different depths of lca.**

order of benefit (line 3). 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 `Hero-Type`). For an equality oracle query, the entity cluster for $v$ is identified if $q_e$ returns `True`. This algorithm is particularly beneficial when the dataset contains a large number 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. First, we show that `Hero` recovers the hierarchy in less than $O(n \log n)$ queries for most realistic settings. Next, we analyze progressiveness of the presented techniques, by considering the processing error and data error models to define the similarity distribution of record pairs. We present proofs in the full version [4].

**Processing Error Model.** According to the processing error model, similarity of a pair of records $(u, v)$ is sampled independently from a normal distribution that depends on the depth of `lca`$(u, v)$. This noise model is motivated by real world datasets, `phylogenetic` (names of animal species with the hierarchy corresponding to their phylogeny) and `camera` (different camera products with the hierarchy corresponding to their brand-model categorization). Figure 6 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 `depth`(`lca`$(u, v)$) $= d$, is sampled independently according to a normal distribution with mean $\mu_d$ and variance $\sigma^2$.

Figure 6 shows that the expected similarity of a pair $(u, v)$ is higher than that of $(u, w)$ if `depth`(`lca`$(u, v)$) $>$ `depth`(`lca`$(u, w)$). Therefore, we assume that $\mu_d > \mu_{d'}$, if $d > d'$. Under this noise model, the ideal similarities correspond to $\sigma = 0$. To analyze the robustness of our techniques, we evaluate a condition on $\sigma^2$ for which our algorithm achieves optimal query complexity.

## 7.1 Query Complexity

This section analyzes the query complexity of Algorithm 8 to construct the hierarchy $H$ over records $V$. First, we analyze the correctness of the candidate hierarchy $\bar{H}$ and then use it to evaluate the query complexity. Let $\mu$ denote the minimum absolute difference in expected similarity of two pairs of nodes. Mathematically, $\mu = \min_{d \neq d'}\{|\mu_d - \mu_{d'}|\}$ and we assume that $\mu$ is constant.

**Initialization.** Algorithm 1 follows a top-down approach by iteratively partitioning $X$ to identify the different branches of the hierarchy. We prove the following result for each iteration.

Lemma 7.2. *For a given collection of records $X$, the partition identified by Algorithm 1 is correct with a probability of $1 - \frac{1}{n^2}$ if*

- $\sigma^2 \leq \mu^2/10$ *if each branch contains $\Omega(\log n)$ leaf level records.*
- $\sigma^2 \leq \mu^2/(10 \log n)$ *otherwise.*

Using this result, we derive the algorithm's query complexity.

Theorem 7.3. *For a collection of records $V$, Algorithm 8 constructs the hierarchy $H^*$ under the processing error model in*

- $O(n \min\{\alpha, \log n\})$ *queries if $\sigma^2 < \mu/(10 \log n)$.*
- $O(n \log n)$ *queries if $\sigma^2 < \frac{\mu}{10}$ and $O(n\alpha \log n)$ otherwise.*

*where $\alpha$ is the maximum degree of an internal node in $H^*$.*

**Data Error.** The processing error model assumes that similarity of every pair of records is sampled independently. However, some records can have higher error due to presence of noisy tokens, missing information or incorrect information. In data error model, we assume that each record can be noisy independently with probability $q' = \Theta(\frac{\log n}{n})$ and all pairwise similarities involving these noisy records are sampled from a different distribution which is characterized based on the type of data error. In case of missing information, the pairwise similarities for noisy node (say $u$) are generated according to a hypothetical hierarchy $H'$, which is equivalent to $H^*$ for all nodes except $u$ and the noisy node $u$ is connected arbitrarily to one of its ancestors. Under this assumption, we show that Algorithm 8 achieves $O(n \log n)$ query complexity.

Theorem 7.4. *For a collection of records $V$, Algorithm 8 constructs the hierarchy $H^*$ in $O(n \log n)$ queries with a probability of $1 - \frac{1}{n}$ if $\sigma^2 < \frac{\mu}{10}$ for processing error and missing information error with probability $q' = O(\log n/n)$.*

**Progressive behavior.** The progressiveness of a technique is measured in terms of the number of identified pairwise relationships. We consider the minimum ratio of F-score at any iteration, $\min F_t^{OPT}/F_t$ to evaluate the progressiveness of a technique [22], where $F_t$ denotes the F-score at any iteration $t$.

Lemma 7.5. *Our algorithm has ideal progressive F-measure under processing error model with* $\sigma^2 < \frac{\mu}{10 \log n}$ *and Insertion sort algorithm [19] has worst case progressiveness approximation of* $O(2^{-n/4+1})$ *for a binary hierarchy.*

## 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 [8, 31, 39–41, 50], phylogeny construction [19], and hierarchical clustering [14, 15, 20]. Much of the prior work on taxonomy construction [8, 31, 39–41, 50] 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 [46]. Recent techniques [46] 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, 15, 20] 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 [7, 10, 12, 28, 48]. 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 [19] studied triplet querying strategies to generate binary hierarchies. 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 [25] 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 [21] 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 [17, 24] for a more detailed survey on ER). In this work, we focus on crowdsourced ER techniques [22, 26, 29, 53–56]. 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.

**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 fairness metrics [34], correlation clustering [49], classification [33, 47], top-$k$ selection [13, 16, 18, 30, 37, 38, 43, 51], information retrieval [36], skyline computation [52] and several other machine learning tasks. Many empirical crowdsourcing studies have shown the ability of crowd members to answer such queries accurately. There has been an independent line of work [9, 57, 60] that 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.

(a) `Amazon` (b) `Phylogenetic`

**Figure 7: Ground truth hierarchy structure. 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 cluster.**

**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 EXPERIMENTS

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

### 9.1 Setup

Before presenting the results, we describe the different datasets, experiment setup and the different baselines considered in the evaluation. As a frame of reference, we plot the ideal curve that achieves the highest progressive F-score.

**Datasets.** We consider six different real-world datasets consisting of hierarchies of varying depth, average degree and shape. Figure 7 shows the difference in shapes of two such hierarchies.

- `Phylogenetic` dataset contains scientific names of bird and insect species along with textual descriptions collected from Wikipedia. The hierarchy corresponds to the phylogenetic evolution tree obtained from [45].
- `DMOZ` [2] is an open-content directory of web pages over the web along with a hierarchy useful to organize these webpages. The dataset contains url and a dump of the text on the web page along with ground truth categorization.
- `Cars` comprises of images of different models of cars. We generate textual descriptions using Google's vision API [3].
- `Camera` [1] is a collection of specifications of cameras collected from over 25 retail companies and the hierarchy corresponds to the brand-model categorization.
- `Amazon` [32] contains descriptions of products and the amazon catalog ontology is used as the ground truth.
- `Geography` dataset [6] contains names of cities across the world and the internal nodes of the hierarchy correspond to the state, country and continent.

**Figure 8: F-score vs Queries comparison.**

**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 [42] 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 [22]. We construct $100 \times 100$ 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 [22, 58] for more details.

**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) `InsSort` considers the extension of the insertion sort algorithm [19] for non-binary hierarchies to generate a hierarchy, followed by `hybrid` [22] to identify equality relationships. (ii) 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 [22] that treats each internal node as a candidate cluster for its benefit calculation. (iii) Average Linkage (denoted by `AverageLink`) is an Agglomerative clustering technique. We used the sklearn package [5] to construct the hierarchy and then run triplet queries bottom up to merge neighboring internal nodes. In addition to these baselines, we plot the ideal curve that corresponds to the ideal strategy aimed at growing clusters in decreasing cluster size and does not ask any wasteful queries. In order to verify insertion of a new branch at any internal node, `ideal` requires $O(\alpha)$ queries, where $\alpha$ is the degree of the internal node. Since some datasets have a very high value of $\alpha$, we use a threshold $\theta = \log n$ to insert the branch. The verification of these branches is performed after all nodes have been processed.

**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

**Figure 9: F-score vs Queries comparison.**

of a record $u$ to the pairwise relationships is the same as that of $v$ whenever $u.\texttt{parent} = v.\texttt{parent}$. 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 $n$.

## 9.2 Result Quality

Figures 8 and 9 compare the F-score of `Hero` with other baselines on multiple datasets. Across all datasets, `Hero` 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 `Hero` to achieve comparable F-score across all large scale datasets. `AverageLink` generates a hierarchical clustering over the records without asking 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`. 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. `Hero` 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, `Hero` 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 `Hero` 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`

(a) Processing error (b) Data error

**Figure 10: Query complexity for varying similarity noise.**

(a) cars (b) dmoz

**Figure 11: Comparison between Hero-Type and Hero**

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 of the 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. Hero achieves near-optimal progressive F-score and performs much better than all other baselines. The overhead in queries for Hero with respect to ideal is due to the extra queries required to identify the correct internal node. Poor performance of Hybrid and InsSort is due to the high degree of internal nodes. For a given internal node (of degree $\alpha$), InsSort queries the branches randomly, which requires $O(\alpha)$ queries to identify the correct branch. On the other hand, 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 Hero is highest for Phylogenetic dataset due to presence of noise in similarity values. To validate the effect of noise, Figure 10 considers synthetic similarities and compares the query complexity with the theoretically proven bounds in Section 7. Figure 10(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 $\sigma^2$. As discussed in Section 7, $\mu$ 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 Hero 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 10(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 are erroneous. In this case, we observe that the query complexity of the technique is directly proportional to the fraction of erroneous nodes.

**Comparison of Hero and Hero-Type.** Hero leverages equality oracle queries to identify clusters of records referring to the same entity and uses triplet queries to identify the hierarchy structure. This

**Table 2: Running Time Comparison**

| Dataset | Hero | 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) cars (b) dmoz

**Figure 12: Varying oracle error**

technique is particularly beneficial for cars and camera datasets, where many records refer to the same entity. Such clusters of records can be efficiently identified using equality oracle queries. Figure 11 shows the benefit of Hero to boost the progressive quality of the hierarchy construction procedure for cars but does not help dmoz due to absence of any pair of records that refer to the same entity.

**Oracle error.** The experiment in Figure 12 assumes that the oracle answers all queries correctly. However, some of the 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* Hero, we leverage the random graph toolkit [23] for each oracle query. Figure 12 shows the impact of noise in oracle answers on the queries required to construct the hierarchy. It shows that Hero 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 [23].

As an additional experiment, we trained a random forest classifier using active learning for cars and camera datasets and used it as an oracle. The classifier was observed to achieve more than 0.9 F-score on individual queries and the final F-score of the hierarchies constructed by Hero-Type was also more than 0.9.

**Running Time.** The running time of Hero to reach 0.90 F-score as compared to other baselines is shown in Table 2. Hero 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. Hero requires less than 2 minutes on phylogenetic dataset and finishes in less than 12 hrs for geography dataset.

## 10 CONCLUSION

In this paper, we formalize the Hierarchical Entity Resolution problem using an Oracle. We propose Hero, an efficient query ordering strategy that leverages pairwise record similarities to prioritize triplet and equality oracle queries. We prove that Hero 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.

# REFERENCES


[1] Alaska benchmark http://alaska.inf.uniroma3.it/.
[2] Dmoz https://dmoz-odp.org/.
[3] Google vision api https://cloud.google.com/vision.
[4] Hierarchical entity resolution using an oracle. https://people.cs.umass.edu/~sainyam/hero_fullversion.pdf.
[5] Scikit-learn https://scikit-learn.org/stable/modules/clustering.html.
[6] World cities database https://simplemaps.com/data/world-cities.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] Daniel G Brown and Jakub Truszkowski. Fast error-tolerant quartet phylogeny algorithms. In *Annual Symposium on Combinatorial Pattern Matching*, pages 147–161. Springer, 2011.
[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] 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.
[16] Susan Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Top-k and clustering with noisy comparisons. *ACM Trans. Database Syst.*, 39(4), December 2015.
[17] Xin Luna Dong and Divesh Srivastava. Big data integration. Morgan & Claypool, 2015.
[18] Eyal Dushkin and Tova Milo. Top-k sorting under partial order information. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1007–1019, 2018.
[19] 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.
[20] 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.
[21] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
[22] Donatella Firmani, Barna Saha, and Divesh Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.
[23] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Robust entity resolution using random graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 3–18, 2018.
[24] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.
[25] 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.
[26] 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.
[27] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
[28] 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.
[29] Anja Gruenheid, Besmira Nushi, Tim Kraska, Wolfgang Gatterbauer, and Donald Kossmann. Fault-tolerant entity resolution with the crowd, 2015.
[30] 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.
[31] 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.
[32] 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.
[33] Max Hopkins, Daniel Kane, Shachar Lovett, and Gaurav Mahajan. Noise-tolerant, reliable active classification with comparison queries. *arXiv preprint arXiv:2001.05497*, 2020.
[34] Christina Ilvento. Metric learning for individual fairness. *arXiv preprint arXiv:1906.00250*, 2019.
[35] Sampath K Kannan, Eugene L Lawler, and Tandy J Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21(1):26–50, 1996.
[36] Ehsan Kazemi, Lin Chen, Sanjoy Dasgupta, and Amin Karbasi. Comparison based learning from weak oracles. *arXiv preprint arXiv:1802.06942*, 2018.
[37] Rolf Klein, Rainer Penninger, Christian Sohler, and David P Woodruff. Tolerant algorithms. In *European Symposium on Algorithms*, pages 736–747. Springer, 2011.
[38] 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 International Conference on Management of Data*, pages 1415–1430, 2017.
[39] 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.
[40] Roberto Navigli, Paola Velardi, and Stefano Faralli. A graph-based algorithm for inducing lexical taxonomies from scratch.
[41] James B Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109–129, 1997.
[42] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9):684–695, 2016.
[43] Vassilis Polychronopoulos, Luca De Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. Human-powered top-k lists. In *WebDB*, pages 25–30, 2013.
[44] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational intelligence*, 9(3):268–299, 1993.
[45] Cristina Roquet, Sébastien Lavergne, and Wilfried Thuiller. One tree to link them all: a phylogenetic dataset for the european tetrapoda. *PLoS currents*, 6.
[46] 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.
[47] 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.
[48] Jakub Truszkowski, Yanqi Hao, and Daniel G Brown. Towards a practical o (n logn) phylogeny algorithm. *Algorithms for molecular biology*, 7(1):32, 2012.
[49] Antti Ukkonen. Crowdsourced correlation clustering with relative distance comparisons. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1117–1122. IEEE, 2017.
[50] Paola Velardi, Stefano Faralli, and Roberto Navigli. Ontolearn reloaded: A graph-based algorithm for taxonomy induction. *Computational Linguistics*, 39(3):665–707, 2013.
[51] 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.
[52] 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.
[53] 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.
[54] Vasilis Verroios, Hector Garcia-Molina, and Yannis Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1133–1148, 2017.
[55] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.
[56] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *arXiv preprint arXiv:1208.1927*, 2012.
[57] Peter Welinder, Steve Branson, Pietro Perona, and Serge Belongie. The multidimensional wisdom of crowds. *Advances in neural information processing systems*, 23:2424–2432, 2010.

[58] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.
[59] 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.
[60] 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.
[61] 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.

# A EXTENSION OF PRIOR STRATEGIES

In this section, we present an extension of prior ER and hierarchical clustering strategies that can be used as querying procedures for Problem 1.

**`InsSort`.** This strategy (Algorithm 11) processes the collection of records $V$ in a random order and maintains a hierarchy $H$ over processed records. In each iteration, a new record $v$ is inserted into the hierarchy containing previously processed records. This insertion step is performed with the help of FindSibling subroutine. The original strategy in [19] is applicable for binary hierarchies but Algorithm 11 works for general hierarchies. The main difference is that a query of a leaf level node with respect to an internal node required a single query in [19], as opposed to $p/2$ queries in our case. Overall, this algorithm has worst case query complexity of $O(np \log n)$, where $p$ is the maximum degree of an internal node. In terms of the three principles of achieving high progressive recall, `InsSort` maintains a single connected hierarchy but does not grow large clusters and does not focus in early discovery of internal nodes.

**Algorithm 11** `InsSort`

**Require:** Hierarchy $H$, records $V = \{u_1, \ldots, u_n\}$
  $V \leftarrow$ `shuffle`$(V)$
  $root \leftarrow$ `new_node()`
  $root \leftarrow root.add_children(V[0])$
  $root \leftarrow root.add_children(V[1])$
  **for** $i = 2, \ldots, n-1$ **do**
    $S \leftarrow$ `get_descendants`$(root)$
    $x \leftarrow$ FindSibling$(V[i], S, H)$
    $x.add_children(V[i])$
  **return** $root$

**`QuickSort`.** The `QuickSort` strategy is a randomized algorithm that chooses two leaf level records as pivots (say $v_1, v_2$) and infers the relative structure of the tree with respect to these pivots with queries of the form $query(u, v_1, v_2), \forall u \in V$. The nodes are then partitioned into four disjoint buckets based on the response to these queries. This procedure is recursively continued to identify the hierarchy structure of the four buckets. These hierarchies are then merged to form the final hierarchy. We refer the reader to [19] for more details. This procedure neither maintains a single connected hierarchy, nor focuses on large clusters.

**ER based techniques.** `Hybrid` technique performs online ER maximizing progressive recall of the resolved entities by growing large clusters of records that refer to same entity. It maintains a set of processed clusters and sequentially processes each record in decreasing order of their expected cluster size. In each iteration, a new record is queried with the identified clusters in non-increasing order of its benefit. The benefit of a record $v$ with respect to cluster $C$ is estimated as the expected gain in recall if $v$ is added to $C$. To extend `hybrid` to hierarchical setting, we consider laminar mapping of each internal node as a separate cluster and query the internal nodes in order of benefit. For more details on benefit calculation and query ordering, please refer [22]. Other prior techniques like [55] and [56] can similarly be extended. However, an extension of [56] queries triples independently based on the relative similarity

of records which may not maintain a single processed hierarchy. Before querying a triple, it invokes `identify_inferred` to make sure that the new query can not be inferred from previous responses.

**Similarity based clustering.** Agglomerative Clustering techniques are widely used in practice to construct binary hierarchies from pairwise similarities. These bottom-up clustering techniques initialize every record in a different leaf level cluster and then recursively merge two closest clusters[6] until all the records have been merged. The different sets of clusters identified across the different iterations of the algorithm correspond to the laminar representation of the hierarchy. These techniques are not robust to noise in similarity values and require $O(n^2)$ running time.

# B INFERENCE

Following this methodology, all queries $q \in Q$ can be represented mathematically in the form of inequality constraints and the hierarchy can be constructed by solving a constraint satisfaction problem (CSP), as shown in Figure 13. The different constraints in this CSP include the inequality constraints for each query $q \in Q$ and triple constraints that for any triple combination of records, atleast two of the three lca's are same and the third lca has higher depth. Such a constraint satisfaction problem can be solved using standard CSP solvers [44]. If there exists a unique solution for the CSP, then all records can be arranged in a single hierarchy. For a given triple query $(x, y, z)$, if the value corresponding variables $\texttt{depth}(\texttt{lca}(x, y))$, $\texttt{depth}(\texttt{lca}(y, z))$ and $\texttt{depth}(\texttt{lca}(x, z))$ are uniquely determined by the CSP solver then the response to $q(x, y, z)$ can be inferred from the set $Q$. All these uniquely determined triple relationships can be represented as a collection of hierarchies $\mathcal{H}$ that compactly represents all inferrable triples. One of the major drawbacks of running a CSP solver is that it requires atleast $\binom{|Q|}{3}$ constraints, which can be inefficient for large values of $Q$.

## B.1 Other Variations

All steps `Hero-Type` and `Hero` do not assume presence of any side information. In the presence of side information about depth of the hierarchy, their query complexity reduces. Consider an ER dataset with no type hierarchy. The cluster representation in this case is same as that of a 2-level hierarchy with records referring to same entity as siblings and the root node has degree $k$, denoting the number of clusters. Let $H$ denote the hierarchy on processed records and $v$ be the record processed in current iteration. An oracle query $(v, v_1, v_2)$ where either $(v, v_1)$ refer to same entity or $(v, v_2)$ refer to same entity is sufficient to insert $v$ into the hierarchy. This is equivalent to an intra-cluster pairwise query in ER. In this case, `Hero-Type` does not query more than one intra-cluster edge to insert the node $v$. With these optimizations in the presence of side information about hierarchy, `Hero-Type` has better query complexity.

## B.2 `Hero`

In addition to these stopping criterion, `Hero` also calculates the likelihood that the dataset contains large clusters of records referring to same entity. This likelihood is calculated by estimating the average number of queries required to insert the $\log n$ recently processed records if `Hero` is not allowed to use use pairwise queries. If the number of triple queries estimated are fewer than the pairwise queries, then it switches to `Hero-Type` and does not use pairwise queries for rest of the iterations. At the end of second phase, `Hero-Type` uses `InsSort` as the final phase.

[6]The distance between any pair of clusters is calculated as the maximum distance between any pair of records between the clusters.

## B.3 Benefit Calculation

Algorithm 13 and 12 present the pseudo code for benefit calculation with respect to triplet and equality queries.

**Algorithm 12** `benefit_triple`

**Require:** Hierarchy $H$, record $v$
  $X \leftarrow H.\texttt{internal_nodes}()$
  **for** $x \in X$ **do**
    $\texttt{benefit}(x, v) = p_{x \to v} \times recall_g(x \to v)$
    $x' \leftarrow$ Internal node corresponding to the partition of $(x, x.parent())$
    $\texttt{benefit}(x', v) = p_{x' \to v} \times recall_g(x' \to v)$
  Sort `benefit` in non-increasing order **return** `benefit`

**Algorithm 13** `benefit_equality`

**Require:** Hierarchy $H$, record $v$
  $X \leftarrow H.\texttt{entity_labelled_nodes}()$
  **for** $x \in X$ **do**
    $\texttt{benefit}(x, v) = p_{x \to v} \times recall_g(x \to v)$
  Sort `benefit` in non-increasing order
  **return** `benefit`

# C THEORETICAL ANALYSIS

In this section, we analyze the query complexity of our algorithm under different noise models, motivated by real world datasets. First, we show that `Hero-Type` recovers the hierarchy in less than $O(n \log n)$ queries for most realistic settings. Next, we analyze progressive behavior of prior techniques, lower bound for an ideal strategy and extension to other settings.

For this analysis, we consider the edge and node error model to define the similarity distribution of record pairs.

**Edge Error Model.** According to the edge noise model, similarity of a pair of records $(u, v)$ with $depth(lca(u, v)) = d$ is sampled independently from a normal distribution with mean $\mu_d$ and variance $\sigma^2$. This noise model is motivated by a real world dataset, `phylogenetic` containing names of different animal species and the hierarchy corresponds to the phylogenetic tree of evolution. Figure 6 shows the similarity distribution of pairs at different depth (after curve fitting) in the hierarchy. More formally,

*Definition C.1 (Similarity distribution).* The similarity of pair $(u, v)$, denoted by $s(u, v)$ with their lca at depth $d$, is sampled independently according to a normal distribution with mean $\mu_d$ and variance $\sigma^2$.

$$
\begin{aligned}
\texttt{depth}(\texttt{lca}(u,v)) &> \texttt{depth}(\texttt{lca}(u,w)), \forall q \in Q \text{ where } q(u,v,w) = w \\
\texttt{lca}(v,w) &= \texttt{lca}(u,w), \forall q \in Q \text{ where } q(u,v,w) = w \\
\texttt{lca}(u,v) &= \texttt{lca}(u,w) = \texttt{lca}(v,w), \forall q \in Q \text{ where } q(u,v,w) = \phi \\
(\texttt{lca}(u,v) &= \texttt{lca}(u,w)) \text{ or } (\texttt{lca}(u,v) = \texttt{lca}(v,w)) \text{ or} \\
(\texttt{lca}(u,w) &= \texttt{lca}(v,w)), \forall u,v,w \in V \\
\text{If } \texttt{lca}(u,v) &= \texttt{lca}(u,w) \text{ then } \texttt{depth}(\texttt{lca}(u,v)) > \texttt{depth}(\texttt{lca}(u,w))
\end{aligned}
$$

**Figure 13: The set of constraints for queries $Q$**

Figure 6 shows that the similarity of a pair of record $(u, v)$ is expected to be higher than the similarity of a pair $(u, w)$ if the lca of $u, v$ is present deeper in the tree as compared to the lca of $u$ and $w$. Therefore, we assume that $\mu_d > \mu_{d'}$, if $d > d'$. Under this noise model, the ideal similarities correspond to the setting where $\sigma = 0$. To better understand the robustness of our techniques, we evaluate a condition on $\sigma^2$ for which our algorithm achieves ideal query complexity.

## C.1 Query Complexity

This section analyzes the query complexity of Algorithm 8 to construct the hierarchy $H$ over records $V$. First, we identify the condition when the prior $H_p$ of Algorithm 8 is identified correctly and then use it to evaluate the query complexity. Let $\mu$ denote the minimum absolute difference in expected similarity of two pairs of nodes. Mathematically, $\mu = \min_{d \neq d'}\{|\mu_d - \mu_{d'}|\}$ and we assume that $\mu$ is constant.

**Prior Construction** Algorithm 1 follows a top-down approach by iteratively partitioning $X$, a subset of records. In each iteration, $X$ corresponds to the set of leaf level records for an internal node (say $x$) in the hierarchy and the partitioning of $X$ corresponds to the leaf level records of each branch i.e. $L(y)$, where $y \in \texttt{children}(x)$ (shown in Figure 4). In this analysis, we assume that the hierarchy is binary and the extension to p-ary hierarchies for any value of $p$ is discussed later.

Theorem C.2. *For a given collection of records $X$, the partition identified by Algorithm 3 is correct with a probability of $1 - \frac{1}{n^2}$ if*

*(1) $\sigma^2 \leq \mu^2/10$ if each branch contains $\Omega(\log n)$ leaf level records.*

*(2) $\sigma^2 \leq \frac{\mu^2}{10 \log n}$ otherwise.*

We analyse the two different cases separately. The first claim considers a specific setting where all the branches of $X$ are assumed to contain $\Omega(\log n)$ leaf level records and is shown in Theorem C.3. Under this assumption, we show that the different branches are identified correctly with a high probability if the variance is less than $\mu^2/10$ (a constant). The second claim operates on internal nodes that have fewer leaf level records and is able to identify all branches correctly if variance is less than $\mu^2/10 \log n$ .

**Analysis for higher levels** For all internal nodes that contain $\Omega(\log n)$ leaf level records, Algorithm 1 uses `construct_partition2` subroutine to identify the different branches. First, Claim 1 shows that the similarity of a pair of records $(u, v)$ is bounded close to the expected value with a probability of $1 - q$ for some small constant $q$. Using Claim 1, we show that the identified category of each edge ($A^+$ or $A^-$) on a sample $S$ is correct independently with a constant probability (Claim 4, 5). The correctness probability of this categorization is boosted by considering transitive agreement of every pair of record in $S$ and two clusters $C_1$ and $C_2$ are identified. Once, we identify a partition of $S$ into two clusters, we assign every other node by comparing $\Theta(\log n)$ triple similarities per node.

Claim 1. *For a pairs $(u, v)$ such that $u, v \in V$ and $depth(lca(u, v)) = d$, if $\sigma^2 \leq \frac{\mu^2}{10}$ then $|s(u, v) - \mu_d| < \mu/2$ with atleast $1 - q$ probability where $q$ is a small constant.*

Proof. According to the noise model, $s(u, v) = N(\mu_d, \sigma^2)$. Therefore,

$$
\begin{aligned}
& Pr[|s(u,v) - \mu_d| > \mu/2] \\
\leq \quad & \frac{\sqrt{2}}{\sqrt{\pi}\mu} e^{-\mu^2/2\sigma^2} \\
\leq \quad & q
\end{aligned}
$$

where $q = \frac{\sqrt{2}}{\sqrt{\pi}\mu} e^{-5}$ is a small constant. □

Let the partition of $X$ according to ground truth hierarchy be $C_L^*$ and $C_R^*$ such that $C_L^* \cup C_R^* = X$ and $\alpha = \min(C_L^*, C_R^*)/|X|$. We consider $\alpha$ is a constant for this analysis and the result extends for any general $\alpha$. In order to prove the correct split of records, Claim 2 shows the lower bound of $count(u, v)$ for a pair $(u, v)$ that corresponds to the same branch $C_L^*$ or $C_R^*$.

Claim 2. *For each pair $(u, v)$ such that $u, v \in S$, if $u$ and $v$ belong to the same branch and $s(u, v) > \mu_d + \mu/2$ then $count(u, v) > ((1 - q)^2\alpha + q)|S'|/2$ with a probability of $1 - \frac{1}{n^4}$*

Proof. The expected count of a pair $(u, v)$ is calculated as follows.

$$
\begin{aligned}
E[count(u,v)] &= \sum_{w \in S'} Pr[s(u,v) > s(u,w), s(v,w)] \\
&\geq \sum_{w \in S' \setminus C(u)} Pr[\mu_d + \mu/2 > s(u,w), s(v,w)] \\
&> (1-q)^2 |S' \setminus C(u)| \\
&> (1-q)^2 \alpha |S|
\end{aligned}
$$

Using Hoeffding's inequality,

$$
count(u,v) \quad > \quad E[count(u,v)] - \beta/2
$$

with a probability of $1 - exp(-\frac{\beta^2}{2|S|})$. Choose $\beta = \alpha|S|$ where $\alpha = (1 - q)^2\eta - q$ and $|S| = \frac{8 \log n}{\alpha^2}$. This implies that with a probability

of $1 - \frac{1}{n^4}$,

$$\begin{aligned} count(u,v) &> E[count(u,v)] - \beta/2 \\ &= ((1-q)^2\eta - q)|S'|/2 \end{aligned}$$

□

Claim 3 calculates the upper bound on the count of a pair $(u, v)$ when $u$ and $v$ belong to different branches.

Claim 3. *For each pair $(u, v)$ such that $u, v \in S$, if $u$ and $v$ belong to the different branches and $s(u, v) < \mu_d + \mu/2$ then $count(u, v) < (((1-q)^2\alpha + q)/2|S'|$ with a probability of $1 - \frac{1}{n^4}$.*

Proof.

$$\begin{aligned} & E[count(u,w)] = \sum_{w \in S'} Pr[s(u,v) > s(u,w), s(v,w)] \\ \leq & \sum_{w \in S' \cap C(u)} Pr[\mu_d + \mu/2 > s(u,w)] \\ & + \sum_{w \in S \setminus C(u)} Pr[\mu_d + \mu/2 > s(v,w)] \\ \leq & |S \cap C(u)|q + q|S \setminus C(u)| \\ < & q|S| \end{aligned}$$

Using Hoeffding's inequality,

$$count(u,v) < E[count(u,v)] + \beta/2$$

with a probability of $1 - exp(-\frac{\beta^2}{2|S|})$. Choose $\beta = \alpha|S|$ where $|\alpha| = (1-q)^2\eta - q$ and $|S| = \frac{8 \log n}{\alpha^2}$. This implies that with a probability of $1 - \frac{1}{n^4}$

$$\begin{aligned} count(u,v) &< E[count(u,v)] + \beta/2 \\ &= ((1-q)^2\eta - q)|S'|/2 \end{aligned}$$

□

Using Claim 2 and 3, we show that each pair is labelled correctly with a constant probability.

Claim 4. *For an edge $(u, v)$, such that $u$ and $v$ belong to the same branch then $(u, v) \in A^+$ with a probability of $1 - \frac{1}{n^2}$ if $s(u, v) > \mu_d + \mu/2$.*

Proof. For a given pair of points $(u, v)$ such that $s(u, v) > \mu_d + \mu/2$ and $u, v$ belong to the same branch, Lemma 2 shows that $count(u, v) > (((1-q)^2(p-1)\alpha + q)/2|S'|$ with a probability of $1 - \frac{1}{n^4}$. This shows that the pair $(u, v)$ is added to correctly with a probability of $1 - \frac{1}{n^4}$. With a union bound over all possible pairs, all such pairs added to $A^+$ correctly with a probability of $1 - \frac{1}{n^2}$. □

Claim 5. *For an edge $(u, v)$, such that $u$ and $v$ belong to the different branch then $(u, v) \in A^-$ with a probability of $1 - \frac{1}{n^2}$ if $s(u, v) < \mu_d + \mu/2$.*

Proof. For a given pair of points $(u, v)$ such that $s(u, v) < \mu_d + \mu/2$ and $u, v$ belong to the different branches, Lemma 3 shows that $count(u, v) < (((1-q)^2(p-1)\alpha + q)/2|S'|$ with a probability of $1 - \frac{1}{n^4}$. This shows that the pair $(u, v)$ is added to correctly with a probability of $1 - \frac{1}{n^4}$. With a union bound over all possible pairs, all such pairs added to $A^-$ correctly with a probability of $1 - \frac{1}{n^2}$. □

Lemma **??** shows that the predicted label of a pair $(u, v)$ is incorrect with probability $\frac{1}{n^4}$ if $s(u, v) > \mu_d + \mu/2$ whenever $u, v$ belong to same branch and $s(u, v) < \mu_d + \mu/2$ whenever $u, v$ belong to different branches. Whenever $u, v$ belong to same branch (using Lemma 1), $s(u, v) \leq \mu_{d+1} - \mu/2$ with a probability $q$. Since $\mu_{d+1} - \mu/2 \geq \mu_d + \mu/2$, $s(u, v) \leq \mu_d + \mu/2$ with probability atmost $q$. Similarly, when $u, v$ belong to different branches, $s(u, v) \geq \mu_d + \mu/2$ with probability atmost $q$. Therefore, each pair $(u, v)$ is placed incorrectly within $A^+$ and $A^-$ with a probability $q + o(1)$ and correct otherwise. In order to boost this probability of correctness, we consider another counting mechanism to calculate the triple agreement. Consider a partition of the edges into $A^+$ and $A^-$.

Claim 6. *Consider a pair $(u, v)$ such that $u$ and $v$ belong to the same branch.* $\mathtt{ACount}(u, v) > (1-\epsilon)(1-q)^2(1-\frac{1}{n^4})$ *with a probability of $1 - \frac{1}{n^4}$ if $|S| > 2 \log n / \left(\frac{1}{2} - (1-q)^2(1 - 1/n^4)\right)^2$.*

Proof.

$$\begin{aligned} & E[\mathtt{ACount}(u,v)] \\ & = E[\sum_{w \in S} 1\{(u,w),(v,w) \in A+ \text{ or } (u,w),(v,w) \in A-\}] \\ > & \sum_{w \in S \cap C(u)} Pr[(u,w) \in A+, (v,w) \in A+] \\ & + \sum_{w \in S \setminus C(u)} Pr[(u,w) \in A-, (v,w) \in A-] \\ > & |S|(1-q)^2(1 - \frac{1}{n^4}) \end{aligned}$$

Using Hoeffding's inequality, $\mathtt{ACount}(u, v) > |S|/2$ with a probability of $1 - exp\left(-2|S|\left(\frac{1}{2} - (1-q)^2(1-1/n^4)\right)^2\right)$ which is $1 - \frac{1}{n^4}$ if $|S| > 2 \log n / \left(\frac{1}{2} - (1-q)^2(1-1/n^4)\right)^2$ □

Claim 7. *Consider a pair $(u, v)$ such that $u$ and $v$ belong to the different branches.* $\mathtt{ACount}(u, v) < |S|/2$ *with a probability of $1 - \frac{1}{n^4}$, if $|S| > 2 \log n / \left(\frac{1}{2} - (1-q)^2(1-1/n^4)\right)^2$.*

Proof.

$$\begin{aligned} & E[\mathtt{ACount}(u,v)] \\ & = E[\sum_{w \in S} 1\{(u,w),(v,w) \in A + or (u,w),(v,w) \in A-\}] \\ < & |S|(1 - (1-q)^2(1 - \frac{1}{n^4})) \end{aligned}$$

Using Hoeffding's inequality, $\mathtt{ACount}(u, v) < |S|/2$ with a probability of $1 - exp\left(-2|S|\left(\frac{1}{2} - (1-q)^2(1-1/n^4)\right)^2\right)$ which is $1 - \frac{1}{n^4}$ if $|S| > 2 \log n / \left(\frac{1}{2} - (1-q)^2(1-1/n^4)\right)^2$ □

The above analysis shows that we correctly identify $\Theta(\log n)$ nodes in each branch with a high probability. Claim 8 shows that all other nodes are assigned correctly with a high probability.

Claim 8. *For a point $w \in X \setminus S$ and two partitions $C_1, C_2 \subseteq S$, $\texttt{AssignCount}(w) > \min(|C_1|, |C_2|)/2$ with a probability of $1 - \frac{1}{n^4}$ if $w \in C_1$.*

Proof. Without loss of generality, suppose $|C_1| < |C_2|$. Using Lemma 10, $s(w, v_j)$ is higher than $s(w, v_k)$ with a probability of more than $(1 - 2q)$. Therefore,

$$E[\texttt{AssignCount}(w)] = E[\sum_{v_j \in C_1} 1\{s(w, v_j) > s(w, v_k)\}]$$
$$|C_1|(1 - 2q)$$

This shows that expected $\texttt{AssignCount}(w)$ is atleast $|C_1|/2$ with a probability of $1 - exp(-2(1/2 - 2q)^2|C_1|)$ which is $1 - \frac{1}{n^4}$ if $|C_1| > 2\log n/(1/2 - 2q)^2$ □

Claim 9. *For a point $w \in X \setminus S$ and two partitions $C_1, C_2 \subseteq S$, $\texttt{AssignCount}(w) < \min(|C_1|, |C_2|)/2$ with a probability of $1 - \frac{1}{n^4}$ if $w \notin C_1$.*

Proof. Without loss of generality, suppose $|C_1| < |C_2|$. Using Lemma 10, $s(w, v_j)$ is higher than $s(w, v_k)$ with a probability of more than $2q$. Therefore,

$$E[\texttt{AssignCount}(w)] = E[\sum_{v_j \in C_1} 1\{s(w, v_j) > s(w, v_k)\}]$$
$$|C_1|(2q)$$

This shows that expected $\texttt{AssignCount}(w)$ is atmost $|C_1|/2$ with a probability of $1 - exp(-2(1/2 - 2q)^2|C_1|)$ which is $1 - \frac{1}{n^4}$ if $|C_1| > 2\log n/(1/2 - 2q)^2$ □

Using these results, we conclude the following theorem about the correctness of prior construction.

Lemma C.3. *For a given collection of records $X$, the partition identified by Algorithm 3 is correct with a probability of $1 - \frac{1}{n^2}$ if $\sigma^2 \leq \mu/10$ and each branch has atleast $\Theta(\log n)$ leaf level records.*

**Smaller branches.** In case of small branches, the constant probability of error in similarity values can lead to erroneous results. Therefore, we require stronger assumptions of $\sigma^2 \leq \frac{\mu^2}{10\log n}$. Under this assumption, we show that the similarity of a pair is bounded close to the expected value with a high probability.

Claim 10. *For a pair $(u, v)$ such that $u, v \in V$ and $depth(lca(u, v)) = d$, if $\sigma^2 \leq \frac{\mu^2}{10\log n}$ then $|s(u, v) - \mu_d| < \mu/2$ with atleast $1 - \frac{1}{n^4}$. probability*

Proof. According to the noise model, $s(u, v) = N(\mu_d, \sigma^2)$. Therefore,

$$\begin{aligned} & Pr[|s(u, v) - \mu_d| > \mu/2] \\ \leq \quad & \frac{\sqrt{2}}{\sqrt{\pi}\mu} e^{-\mu^2/2\sigma^2} \\ \leq \quad & \frac{1}{n^4} \end{aligned}$$

□

Using Lemma 10, we show that with a high probability, similarity of a pair that has deeper lca is higher than that of a pair with lower depth of lca.

Claim 11. *For any triple $u, v, w$ such that $depth(lca(u, v)) > depth(lca(u, w))$ then $s(u, v) > s(u, w)$ with a probability of atleast $1 - \frac{1}{n}$*

Proof. Suppose $depth(lca(u, v)) = d$ and $depth(lca(u, v)) = d'$. Since $d > d'$, $\mu_d - \mu_{d'} \geq \mu$. According to Lemma 10, $s(u, v) > \mu_d + \mu/2$ and $s(u, w) < \mu_{d'} - \mu/2$ with a parobability of $1 - \frac{2}{n^4}$. Therefore,

$$\begin{aligned} s(u, v) - s(u, w) &> \mu_d - \mu_{d'} - \mu \\ &= 0 \end{aligned}$$

This shows that $s(u, v) > s(u, w)$ with a probability of $1 - \frac{2}{n^4}$. With a union bound over all triples $u, v, w \in V$, it holds with a probability of atleast $1 - \frac{1}{n}$. □

Claim 12 uses the helper lemma to show that for a given collection of nodes $X$, the $p$ partitions identified correspond to the leaf records in the $p$ different branches corresponding to the internal node.

Claim 12. *For a given collection of nodes $X \subseteq V$, Algorithm 1 identifies the $p$ seed nodes correctly with a probability of $1 - \frac{1}{n}$*

Proof. According to Lemma 11, the similarity of a point $w$ which belongs to different branch has lower similarity than any pair that is present in any of the identified branches with a high probability. Therefore, the farthest point identified by Algorithm 2 in any iteration does not belong to any of the identified branches. □

Lemma C.4. *The hierarchy $H$ returned by Algorithm 1 is same as $H^*$ with a probability of $1 - \frac{1}{n}$ if $\sigma^2 \leq \frac{\mu^2}{10\log n}$.*

Proof. For a given set $X$, Lemma 12 shows that the seed nodes identified belong to different branches with a high probability. Additionally, Lemma 11 shows that all the records are assigned correctly. Therefore, the hierarchy $H$ is identified correctly with a high probability. □

**Discussion.** Theorem C.3 shows that the binary hierarchy constructed is able to recover all branches of size more than $\Theta(\log n)$ correctly with a high probability. To extend the similarity model for p-ary tree, we consider similar similarity distribution with a difference that the expected value of similarity is different for each pair of branch. For example, an internal node electronics has children branches containing camera, monitor, keyboard, etc. The similarity of a camera and monitor is lower than that of keyboard and monitor even though the lca of all the three records is at the same depth from the root. In this setting, Algorithm 1 first constructs a binary hierarchy such that all the internal nodes that are closer to the root in $H^*$ are present closer to the root in $H_p$. This shows that the only difference in $H_p$ and $H^*$ is the presence of extra internal nodes which need to be merged. To perform this merge operation, we consider similarity to probability of all the triples spanning these internal nodes. Given the correctness of similarity to probability, all these levels are merged correctly and $H_p$ corresponds to $H^*$ for

all levels containing more than $\log n$ leaf level records. Therefore, Theorem C.4 extends for p-ary hierarchies.

**Query Complexity** Using the previous analysis we show the following results about query complexity.

THEOREM C.5. *For a collection of records $V$, Algorithm 8 constructs the hierarchy $H^*$ in*

- $O(n \min\{p, \log n\})$ *queries if* $\sigma^2 < \frac{\mu}{10 \log n}$.
- $O(n \log n)$ *queries if* $\sigma^2 < \frac{\mu}{10}$.
- $O(np \log n)$ *queries otherwise.*

PROOF. According to Lemma C.4, the prior $H_p$ constructed in line of Algorithm 8 is correct with a probability of $1 - \frac{1}{n}$ if $\sigma^2 < \frac{\mu^2}{10 \log n}$. Therefore, each node is queried with respect to its true internal node. However, our querying mechanism requires atleast $\min\{p, \log n\}$ queries to add a new branch. This shows that each node is inserted using atmost $(\min\{p, \log n\})$ queries, yielding overall complexity of $O(n \min\{p, \log n\})$.

According to Theorem C.3, the prior hierarchy $H_p$ has all levels correctly identified if the branch contains atleast $\Theta(\log n)$ leaf level records. This shows that the prior hierarchy identified all internal nodes correctly until the branch size reduced to $\log n$. Therefore, in the worst case every new node is queried with all the incorrect internal nodes, yielding a query complexity of $\log n$ to insert it correctly. This shows that Algorithm 8 has a query complexity of $n \log n$. □

## C.2 Extension to node error model

The edge error model assumed that similarity of every pair of record is sampled independently according to the normal distribution. However, some nodes can have higher error due to presence of noisy tokens, missing information or incorrect information. We assume that each record can be noisy with respect to node error independently with probability $q' = \Theta(\frac{\log n}{n})$ and all pairwise similarities involving these noisy records are sampled from a different distribution which is characterized based on the type of node error. In case of missing information error, the pairwise similarities for noisy node (say $u$) are generated according to a hypothetical hierarchy $H'$, which is equivalent to $H^*$ for all nodes except $u$ and the noisy node $u$ is connected arbitrarily to one of its ancestors. Under this assumption, we show that Algorithm 8 achieves $O(n \log n)$.

THEOREM C.6. *For a collection of records $V$, Algorithm 8 constructs the hierarchy $H^*$ in $O(n \log n)$ queries with a probability of $1 - \frac{1}{n}$ if $\sigma^2 < \frac{\mu}{10}$ and missing information error with probability $q' = O(\log n/n)$.*

PROOF. Let $X_u$ denote a random variable which is 1 if $u$ possess node error and 0 otherwise. Let $q' = \frac{a \log n}{n}$ where $a > 1$ is a constant.

$$E\left[\sum_{u \in V} X_u\right] = q'n = a \log n$$

Using Chernoff bound, we can show that $\sum_{u \in V} X_u \geq 3a \log n$ with a probability atmost $e^{-a \log n} < \frac{1}{n}$. According to Theorem **??**, all nodes that obey edge error are attached within $O(n \log n)$ queries. On the other hand, all nodes that possess node error may require $n$ queries per node in the worst case. Therefore, our algorithm has overall query complexity of $O(n \log n)$. □

The extra information node error leads to an increase in similarity of certain record pairs. One of the main reasons for this noise is due to the presence of features that are not used in hierarchical categorization. For example, color of a product in amazon category hierarchy. In such a setting, the pairwise similarities for noisy node (say $u$) are generated according to a hypothetical hierarchy $H'$, which is equivalent to $H^*$ for all nodes except $u$ and the noisy node $u$ is connected arbitrarily to one of the descendants of its parent (any sibling branch). Following the proof of Theorem C.6, we get the following theorem.

THEOREM C.7. *For a collection of records $V$, Algorithm 8 constructs the hierarchy $H^*$ in $O(n \log n)$ queries with a high probability if $\sigma^2 < \frac{\mu}{10}$ and extra information type node error with probability $q' = O(\log n/n)$.*

## C.3 Progressive behavior

The progressiveness of a technique is measured in terms of the number of identified pairwise relationships. We consider the minimum ratio of recall at any iteration, $\min \frac{F_t^{OPT}}{F_t}$ to evaluate the progressiveness of a technique [22], where $F_t$ denotes the recall at any iteration.

LEMMA C.8. *Insertion sort algorithm [19] has worst case approximation of $O(2^{-n/4+1})$ under progressive recall for a binary hierarchy.*

PROOF. Consider a binary hierarchy $H$ such that the two branches of root node contains $n/2$ records each and both the branches have depth $n/2$. Since insertion sort considers nodes in a random ordering, after $n/2$, the expected depth of the constructed hierarchy is $n/4$.

However, the optimal solution constructs one branch of depth $n/2$ completely. Assuming that the contribution of each pairwise relationship decreases exponentially, the maximum contribution of any pair in insertion sort is $2^{-n/4}$ as compared to $2^{-1}$ for the ideal strategy. □

LEMMA C.9. *Our algorithm has ideal progressive F-measure under edge error model with $\sigma^2 < \frac{\mu}{10 \log n}$.*

PROOF. Following the proof of Theorem C.5, the similarity to probability mapping of a triple is accurate with a high probability. Therefore, the benefit ordering chosen by `Hero-Type` is accurate and it queries all records with the correct internal node if $\sigma^2 < \frac{\mu}{10 \log n}$. Therefore `Hero-Type` achieves optimal progressive F-score if $\sigma^2 < \frac{\mu}{10 \log n}$. □

**Error Analysis** The above presented results are proved for a setting where the oracle answers are correct. However, in the presence of error in oracle answers, we use the error correction toolkit [23] that adds a redundancy of $O(\log n)$ queries. In such a setting, `IdentifyBranch`(u,v) requires $O(p \log n)$ queries in the wost case, with $\log n$ random queries for each branch.

## C.4 `Hero` query complexity

In this section, we extend the analysis of Algorithm 8 to bound the query complexity of `Hero` that grows same entity clusters using pairwise queries and constructs hierarchical structure with triple queries. Note that $p$ corresponds to the degree of internal node in the type hierarchy.

Theorem C.10. *For a collection of records $V$, Algorithm 8 constructs the hierarchy $H^*$ in*

- $O(n \min\{p, \log n\})$ *queries if* $\sigma^2 < \frac{\mu}{10 \log n}$.
- $O(n \log n)$ *queries if* $\sigma^2 < \frac{\mu}{10}$.
- $O(np \log n)$ *queries otherwise.*

Proof. According to [22], `Hybrid` inserts each record $v$ into $O(\log n)$ high benefit clusters. If unsuccessful, $v$ is inserted as a singleton cluster. Therefore, the additional overhead of performing pairwise queries is less than $O(n \log n)$ and following the proof of Theorem C.5, we get the desired query complexity. □

## C.5 Hardness

This section calculates the minimum number of queries required to construct the hierarchy $H$. Lemma **??** shows the minimum number of queries required to identify the hierarchy is $\sum \binom{d_i}{2}$, where $d_i$ is the number of children of each internal node $i$ in the hierarchy $H^*$. First, we show that the number of queries required to identify an internal node is atleast $\Theta(|x.\texttt{children}|^2)$.

Lemma C.11. *For any internal node $x$, there should be atleast $\Theta(|x.\texttt{children}|^2)$ edges across the different branches.*

Proof. Consider an internal node $x$ in the hierarchy $H$, let $S$ denote the set of children(x). Consider $b_1 \in S$ and let $L(b_1)$ correspond to the set of leaves of the subtree rooted at $b_1$. Consider all triples $T = \{(u, v, w)$ such that $\texttt{lca}(u, v) = \texttt{lca}(u, w) = \texttt{lca}(v, w) = x, u \in L(b_1)\}$. Since lca of any pair of records is $x$, the three records $u$,$v$ and $w$ belong to different subtrees in $H(x)$. Now, we will show $|T| \geq (|S| - 1)/2$ by contradiction. If $|T| < (|S| - 1)/2$, there exists atleast one branch $b_2 \in S$ such that there is no leaf level record among $L(b_2)$ that belongs to $T$. Therefore, an adversary can place $b_1$ and $b_2$ under a common parent which is a child of $x$. This shows that the hierarchy is not uniquely identified if $|T| < (|x.children| - 1)/2$. Extending this to every branch $b_i \in children(x)$, the minimum number of queries required to identify all the branches of $x$ is $\Theta(|x.\texttt{children}|^2)$ □

Lemma C.12. *For a hierarchy $\mathcal{H}$, any algorithm that does not make $\Omega(\sum_{u \in internal(\mathcal{H})} |children(u)|^2)$ queries will never identify the correct hierarchy.*

Proof. Using Lemma C.11. □

Corollary C.13. *For a hierarchy where all internal nodes have $p$ children, any algorithm requires atleast $\Theta(np)$ queries to identify the correct hierarchy.*

For a special case when the depth of the hierarchy is known apriori, the argument of Lemma **??** does not hold for the (leaf-1) level internal nodes. For all (leaf-1) level internal nodes (say $A$), one query for each leaf level record is sufficient. This yields an overall complexity of $n + \Omega(\sum_{u \in internal(\mathcal{H}) \setminus A} |children(u)|^2)$.

**Insertion sort algorithm** The Insertion sort algorithm has an overall complexity of $np \log n$ and we show that every record requires $O(p \log n)$ queries to be inserted into the hierarchy.

Lemma C.14. *Insertion sort++[19] requires $O(p \log n)$ queries to insert a node into a hierarchy $H$.*

Proof. Consider the new vertex which is being inserted by $v \notin H$. The insertion sort algorithm iteratively identifies an internal node (say $u$) which partitions the tree into various components of smaller size. After selecting the internal node, we require atmost $\lceil p/2 \rceil$ queries to identify the correct branch where $v$ needs to be inserted. According to [19], there exists a separator tree for each hierarchy such that the depth of the separator tree is atmost $\Theta(\log n)$. □

Using this lemma, we get the following result.

Theorem C.15. *Algorithm 11 has an overall query complexity of $O(np \log n)$.*