Hierarchical Entity Resolution using Oracle

Group:
Jiyang Chen(A20490982)
Yang Li(A20478590)
Mingyuan Wang(A20501460)

Content:

Hierarchical Entity Parsing (HEP) is a Natural Language Processing (NLP) technique for identifying and parsing entities with a hierarchical structure from text. This technique is often used to process complex information where there are nested or hierarchical relationships between entities. For example, when working with legal documents, medical records or organisational structured data, it may be necessary to identify and understand various hierarchical entities and their interrelationships. This thesis focuses on hierarchy identification, entity relationship resolution techniques

Abstract:

HierER is a new hierarchical entity resolution methodology that formulates a novel hierarchical ER task by considering the weak supervision of ORACLE forms, which solves the problem of automatically identifying such relationships incorrectly due to noise and heterogeneous representations of records from different sources, as well as solving the series of problems of manually maintaining these relationships that do not scale up to large datasets, HierER leverages triple comparisons of Oracle queries using similarity of pairs of records to minimize oracle query counts and at the same time maximise the hierarchical structure identified.

Introduction:

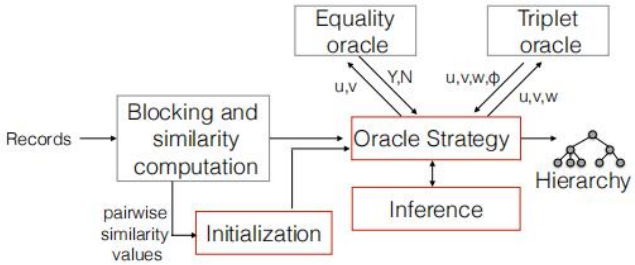
Entity Resolution (ER) Some previous ER techniques use type information of entities to improve ER classification but do not study hierarchically arranged record type recognition, hierarchical clustering methods ignore entity resolution, and these techniques build only binary hierarchies,

so entity resolution (ER) and hierarchical clustering alone cannot solve the existing problem. However, using the two processes of running entity resolution (ER) and then running hierarchical clustering in a pipelined time complexity of $O(n^2)$ is not an ideal solution.

Methodology:

Hier-Type:efficiently handles purely typed hierarchies with arbitrary degree distributions (i.e., all entities are of size 1) and outperforms hierarchical clustering methods with a time complexity of $O(n \log n)$

HierER :solves the hierarchical ER problem and outperforms the pipelined approach (ER plus hierarchical clustering, i.e., either order, with a time complexity of $O(n \log n)$



An overview of the workflow for dealing with layered ER issues is shown in the figure below
The main algorithms:

Algorithm 1 generate-similarity-hierarchy

Require: records X, similarity s, degree α
if $|X| == 1$ **then return** X
$\tilde{H}_X \leftarrow$ new-node()
if $|X| \leq \log n$ **then**
  $\mathcal{P} \leftarrow$ greedy-partition(X, s, α)
else
  $\mathcal{P} \leftarrow$ robust-partition(X, s, α)
for $C \in \mathcal{P}$ **do**
  $\tilde{H}_c \leftarrow$ generate-similarity-hierarchy(C)
  $\tilde{H}_X$.add-children$(\tilde{H}_C)$
$\tilde{H}_X \leftarrow$ merge-internal$(\tilde{H}_X)$
return $\tilde{H}_X$

Algorithm 2 robust-partition

Require: records X, similarity s, degree α
Select subsets $S, S' \subseteq X$ randomly such that $|S| = |S'| = \Theta(\log n)$
for $v_1 \in S$ **do**
  for $v_2 \in S \setminus \{v_1\}$ **do**
    $Count(v_1, v_2) \leftarrow \sum_{v_3 \in S'} \mathbb{1}\{s(v_1, v_2) > s(v_1, v_3), s(v_2, v_3)\}$
    if $Count(v_1, v_2) > |S'|/(2\alpha)$ **then**
      $A^+ \leftarrow A^+ \cup \{(v_1, v_2)\}$
    else
      $A^- \leftarrow A^- \cup \{(v_1, v_2)\}$
$C \leftarrow \phi$
while $|C| < \alpha$ **do**
  $C' \leftarrow$ select a random record from S (say v_1)
  for $v_2 \in S$ **do**
    ACount$(v_1, v_2) = \sum_{v_3 \in S} \mathbb{1}\{(v_1, v_3)$ and $(v_2, v_3) \in A^+$ or $A^-\}$
    if ACount$(v_1, v_2) > |S|/2$ **then**
      $C' \leftarrow C' \cup \{v_2\}$
  $S \leftarrow S \setminus C', C \leftarrow C \cup C'$
$\beta \leftarrow \min_{C_i \in C} |C_i|$
for $v_3 \in X \setminus \cup_{C_i \in C} C_i$ **do**
  assign-count$(v_3, C_i) = \sum_{t \leq \beta} \mathbb{1}\{s(v_3, C_i[t]) > s(v_3, C_j[t]), \forall j \neq i\}$
  $C_\eta \leftarrow \arg\max_{C_i \in C}$ assign-count(v_3, C_i)
  $C_\eta \leftarrow C_\eta \cup \{v_3\}$
return C

Algorithm 3 Hier-Type

Require: records V, seed hierarchy H_T, similarity s
$\tilde{H} \leftarrow$ generate-similarity-hierarchy(V)
$L \leftarrow$ get-expected-cluster-size(V)
$P \leftarrow$ leaves$(H_T), P_t \leftarrow \phi, H \leftarrow H_T$
for $v \in L$ **do**
  $H, P, P_t \leftarrow$ Hier-TypeInsertion$(v, H, P, P_t, \tilde{H})$
for $v \in P_t$ **do**
  $root \leftarrow u$ such that $u \in H \setminus P_t, u = v.ancestor()$
  $H \leftarrow$ find-sibling$(v, root)$
return H

Algorithm 4 Hier-TypeInsertion

Require: record v, Hierarchy H, Processed records P, P_t, candidate hierarchy $\tilde{H}$
$B \leftarrow$ benefit-triplet(v, H, P)
$i \leftarrow 1$
$S \leftarrow initialize - candidates(H)$
for $b \in B \cap S$ **do**
  $o, t \leftarrow$ identify-branch$(v, b, \tau - i)$
  $i \leftarrow i + t$
  $S \leftarrow update(S, o)$
  if $|S| = 1$ **then**
    Insert b as a sibling of S
    break
  if $i \geq \tau$ **then**
    Attach v to the candidate with lowest depth
    $P_t \leftarrow P_t \cup \{v\}$
    return H, P, P_t
    break
  if $i \geq \gamma$ **then**
    $u \leftarrow$ find-sibling(v, S, H)
    break
$H \leftarrow$ expand-branch$(v, H, \tilde{H})$
$P \leftarrow P \cup \{v\}$
return H, P, P_t

Generate-similarity-hierarchy algorithm:

The main objective of the algorithm is to construct a candidate hierarchy from the similarity values which captures the type relationships and provides an infrastructure for the subsequent entity resolution process.The algorithm employs a greedy strategy to quickly partition the records into subsets based on similarity when the record set is small and a more robust approach to partitioning when the record set is large.For each subset in the result of the partitioning, the subtypes are then recursively determined for each subset.The purpose of this approach is to ensure that the type relationships are captured correctly based on the given similarity values.Finally, the internal merging is done in order to simplify and optimise the hierarchical structure

Robust-partition algorithm:

The algorithm is a robust partitioning algorithm which aims to partition records accurately in the presence of noisy similarity values.The key to the algorithm is that it does not just rely on a single similarity value for partitioning but considers the similarity between multiple pairs of records and the consistency between these similarities for robust partitioning in the presence of noise.The algorithm randomly selects two subsets of equal size from the original dataset, the size is proportional to the number of pairs in the original dataset.For each pair of elements in the subset $(v1, v2)$ calculate their similarity to another pair $(v1, v2)$ .The algorithm randomly selects two subsets of equal size from the original dataset, the size of which is proportional to the number of pairs in the original dataset, for each pair of elements in the subset $(v1, v2)$ calculate their similarity to the elements in the other subset and classify them accordingly, based on the results of similarity calculations, the pairs of elements are classified into the high similarity set A^+ or the low similarity set A^-, and the clustering set is constructed by selecting the random elements through iteration and according to similarity The clusters are constructed by iteratively selecting random elements and assigning them to the most suitable clusters according to their similarity with the existing clusters until the preset number of clusters α is satisfied, and then return the constructed clusters.

Hier-Type Algorithm:

The main purpose of this algorithm is to construct an initial hierarchy.The initial hierarchy provided by Hier-Type provides a starting point for subsequent HierER algorithms.The HierER algorithm then further refines and adapts this hierarchy to more accurately reflect the

relationships between entities.The algorithm generates a similarity-based based on the records V The algorithm generates a similarity-based hierarchy ^{-}H based on the records V , calculates the cluster size that the records V should have and stores it in L . Starting with the leaf nodes of the seed hierarchy HT , it initialises the set of processed nodes P and a temporary set of nodes Pt , and the initial hierarchy structure is HT , for each leaf node in L , it updates the hierarchy H , the set P , and the temporary set Pt by using the Hier-Type Insertion function to integrate the leaf nodes into the hierarchy ^{-}H , and the leaf nodes into the hierarchy ^{-}H . nodes into Hierarchy ^{-}H ,for each node V in the temporary set Pt , find the root node in Hierarchy H that is not in Pt but is an ancestor of V ,for each such node V , use the find-sibling function to find and adjust its relative position to its sibling nodes based on similarity relationships.

Hier-Type Insertion Algorithm:

A method for inserting a single record V into an existing hierarchy H , taking into account the set of processed records P and Pt , as well as the candidate hierarchy ^{-}H . The algorithm provides a careful way of integrating the new record into a complex hierarchy while ensuring structural consistency and maximising efficiency.

HierER Algorithm:

the process of HierER is similar to Hier-Type, with the difference that it computes the benefits of v for ternary queries (same as Hier-Type) and equal queries. Then, a sorted list of all candidate positions is sorted in non-increasing order of benefit to insert v . Records v are queried in increasing order of benefit until one of the stopping conditions (the same as in Hier-Type) is reached. For equal Oracle queries, if q_e returns True, the cluster of entities for v is identified. This algorithm is particularly effective when the dataset contains a large number of records pointing to the same entity.

Blocking and Similarity:

Using standard blocking, which generates a block for each token in the input record set, reduces the number of pairs considered when calculating similarity, similarity maps values to probability distributions using calibration methods for equal and triple oracle queries respectively.

Initialization:

The Hier-Type algorithm is invoked to generate a candidate type hierarchy layer, which is used to guide the downstream query strategy

Oracle strategies：For the candidate hierarchies generated by the initialisation, a higher priority is given to queries that produce higher F-score increments for the reasoning engine to prioritise Oracle queries.

Inference:

Calling the HierER algorithm

Main Findings:

Under moderate noise conditions, both Hier-Type and HierER require $O(n \log n)$ oracle queries and are likely to obtain the best asymptotic F-score. Experimental results show that Hier-Type and HierER scale to datasets of millions of records and outperform the benchmark solution in both efficiency and effectiveness. The query complexity and asymptotics of our algorithms are analysed for different similarity noise models, and in most practical cases, HierER requires less than $O(n \log n)$ of query time to construct the correct hierarchy

Discussion and Analysis:

- Algorithm performance analysis of HierER algorithm under different similarity noise models(Theoretical part)：

Normally Distributed Similarity: the article states that the similarity of record pairs is usually normally distributed with its mean varying according to the depth of the lowest common ancestor (lca) in the hierarchy. This means that the similarity between data points varies according to their relative position in the hierarchy.

Independent Edge/Handling Errors : ρ < 0.5, indicating that the proportion of noisy record pairs is less than 50%. σ^2 < μ/10: Here σ^2 denotes the variance of the noise, while μ is the minimum average difference in the similarity of the record pairs. This condition implies that the variability of the noise is small relative to the average difference in similarity. The algorithm HierER is constructed $O(n \log n/(1-2\rho)^2)$ with probability 1 - $\frac{1}{n}$ in the H^*

System data error: ρ < 0.5,denotes the proportion of leaf nodes affected by noise is less than 50%, and α denotes the proportion of randomly connected leaf nodes in the noisy hierarchical structure H . When $\alpha = \Theta(\log n/n)$, then the HierER algorithm will construct the correct hierarchical structure H^* with probability 1- $\frac{1}{n}$ in $O(n \log n)$ ternary comparisons

- Analysis of the effectiveness of the HierER algorithm (Experimental part):

Evaluation metric: The F-score is a popular metric for hierarchical ER, where we compare the performance of various techniques by measuring asymptotic F-scores

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

Pipeline Baseline Strategy:

AverageLink: sklean cluster clustering technique + Triplet oracle merge neighboring internal nodes

HiExpan: task-guided taxonomy construction via hierarchical tree expansion, a technique that uses an initial seed hierarchy to generate additional internal nodes and assumes that all internal nodes in the hierarchy can be accessed; internal nodes are added to run the algorithm.

InsSort: Insertion sort algorithm extended with a non-binary hierarchy to generate hierarchies and then use a hybrid algorithm to recognize equivalence relations

Hybrid: an adaptation of the entity resolution strategy using an algorithm that seeks to maximize asymptotic recall, which treats each internal node as a candidate cluster to compute its gain.

HierTr.Eq: a sequential pipeline that generates a type hierarchy (Hier-Type) using triple queries and then identifies entity nodes using equality queries

HierTr: is a hierarchical algorithm that uses only triple queries to build type hierarchies using hierarchical types without recognizing entity nodes

HierER-Nb: Same as HierER but without benefit calculation at query time. It processes records in randomized order.

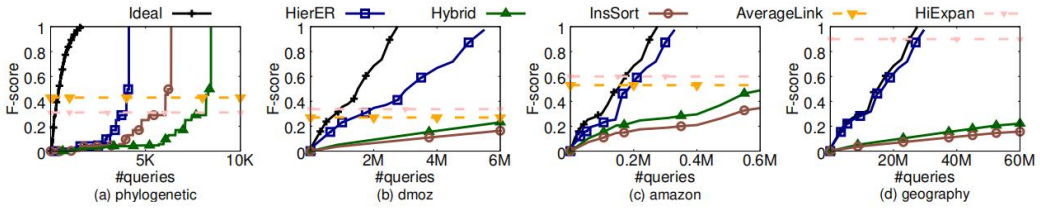


Figure 10: F-score vs #queries comparison for datasets where all records refer to distinct entities. In these datasets, `HierER` performs similar to its pipelined variants `HierTr.Eq` and `HierTr` since equality oracle does not provide any information.

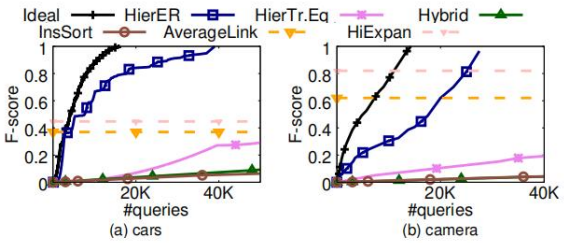


Figure 11: F-score vs #queries comparison for datasets with high-degree entity nodes.

Quality of results:

Figures 10 and 11 compare the F-score of HierER with the other baselines on multiple datasets.The HierER asymptotic F-score scores are the highest and closest to the ideal curve on all datasets. Due to the high level of noise in the similarity values, HierER is unable to recognize the presence of many internal nodes in the initial stages of parsing. This delay in recognizing all internal nodes of the hierarchy affects the asymptotics. Nevertheless, the overall complexity of HierER is about $O(n \log n)$

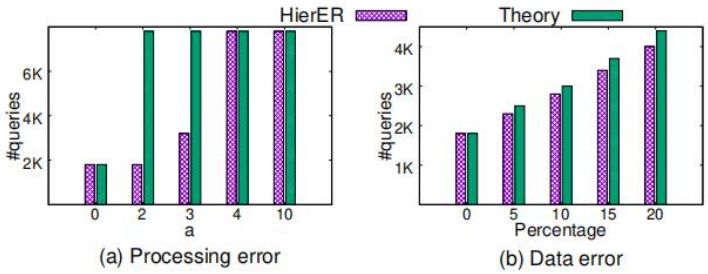


Figure 12: Query complexity for varying similarity noise.

Comparison with theoretical results:

Due to the presence of noise in the similarity values, the gap between the ideal value and HierER is largest in the phylogenetic dataset,the query complexity increases when the noise is larger,but stabilizes at $O(n \log n)$, the query complexity is proportional to the error

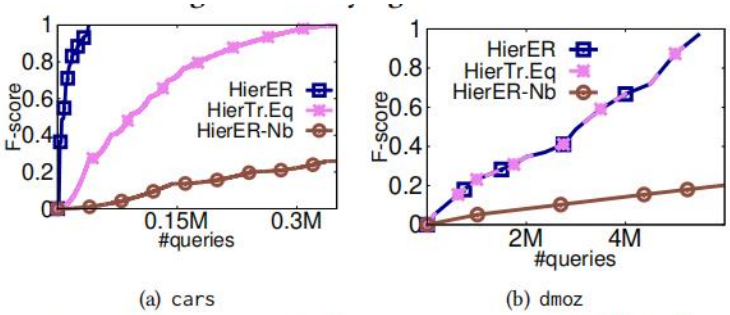


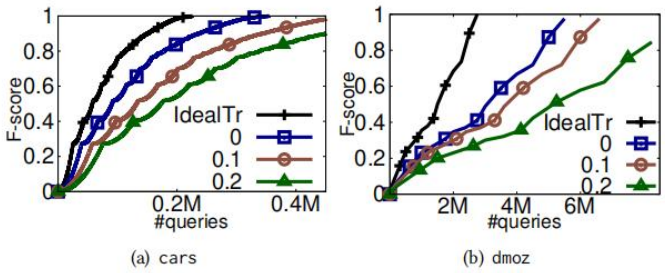
Figure 14: Comparison between variants of HierER

Query Strategy Variants:

HierER outperforms the other query strategy variants for many records in the dataset all pointing to the same entity, and for no pair of records pointing to the same entity, HierER is comparable to the HierTr.Eq asymptotic F-score

Oracle error:

For errors less than 0.3, HierTr correctly recognizes hierarchies and asymptotic F-scores outperform HierER, while query complexity increases by $O(\log n)$ due to the introduction of redundancy



(a) cars (b) dmoz

Figure 13: Varying oracle error

Runtime:

Runtime is linearly related to the number of queries required, so for most large-scale datasets, HierER's runtime is shorter than other techniques

Table 2: Running Time Comparison

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

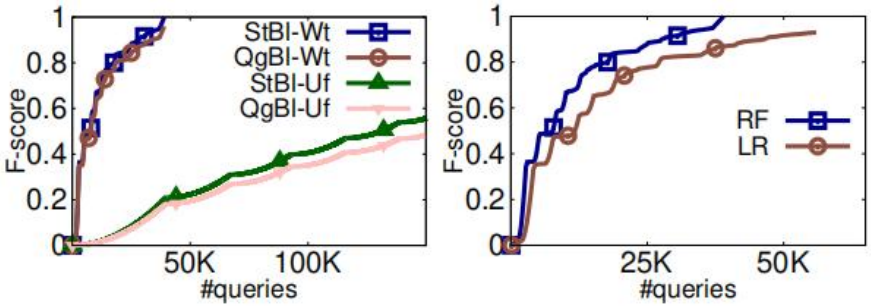
Analysis of the effectiveness of HierER default settings:

Blocking:

meta-blocking ,StBl-Wt denotes; Q-gram based blocking with TF-IDF weights (QgBl-Wt); Q-gram based blocking with unity weights (QgBl-Uf); standard blocking with unity weights (StBl-Uf), HierER default performance of meta-blocking Best

Classification:

HierER's Default Settings Outperform Logistic Regression (LR) Using Random Forest Classifier (RF)



(a) Blocking (b) Classifier

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

Objective Critique:

1. Although HierER uses a robust partitioning method to reduce the computational effort, for million-size datasets it does not achieve the O(nlogn) ideal complexity, so its computational complexity may still be an issue on large-scale datasets.
2. Choice of Hierarchy: The performance of HierER also depends on the hierarchy of entities. An ill-conceived hierarchy may lead to performance degradation as well as error rates.

Conclusion:

HierER is a novel hierarchical entity resolution approach that exploits the hierarchical structure between entities to improve the efficiency and accuracy of entity resolution, and it utilizes an efficient query ranking strategy that exploits the similarity of pairs of records to prioritize ternary and equal Oracle queries. We show that HierER can $O(n \log n)$ construct basic truth hierarchies in queries under reasonable assumptions of processing and data error modeling. We empirically demonstrate its effectiveness with various real datasets

References：