Node ranking in labeled networks*

Chamalee Wickrama Arachchi, Nikolaj Tatti †

Abstract

The entities in directed networks arising from real-world interactions are often naturally organized under some hierarchical structure. Given a directed, weighted, graph with edges and node labels, we introduce ranking problem where the obtained hierarchy should be described using node labels. Such method has the advantage to not only rank the nodes but also provide an explanation for such ranking. To this end, we define a binary tree called label tree, where each leaf represents a rank and each non-leaf contains a single label, which is then used to partition, and consequently, rank the nodes in the input graph. We measure the quality of trees using agony score, a penalty score that penalizes the edges from higher ranks to lower ranks based on the severity of the violation. We show that the problem is **NP**-hard, and even inapproximable if we limit the size of the label tree. Therefore, we resort to heuristics, and design a divide-and-conquer algorithm which runs in $\mathcal{O}((n+m)\log n + \ell R)$, where R is the number of node-label pairs in the given graph, ℓ is the number of nodes in the resulting label tree, and n and m denote the number of nodes and edges respectively. We also report an experimental study that shows that our algorithm can be applied to large networks, that it can find ground truth in synthetic datasets, and can produce explainable hierarchies in real-world datasets.

1 Introduction

Many real-world networks naturally inherit a hierarchical structure. Often times, these interactions among the entities in the network, provide the means of finding underlying hierarchical organization of the network. Examples include ranking teams in a football league [14], exploring rankings in social networks [8, 12], terrorist networks [13], and animal networks [9].

The problem of ranking an individual, based on its interactions with others, has drawn attention over past decades [8, 14, 19]. More formally, given a graph G we are looking to assign each node i, a rank $r(i)$, an integer that minimizes a penalty. Here the score, called *agony*, penalizes the backward edges (u, v) based on the difference between the ranks $r(u) - r(v)$.

Many practical networks contain information beyond the network structure. Such labels can be used in numerous applications such as community detection and clustering [1, 2, 6, 7, 15]. Here, the labels are used to explain the obtained results, which ideally increase the usability of these results to the practitioners.

In this paper, we propose an approach to infer *explainable* hierarchies in labeled, weighted, directed networks. More specifically, we look for a decision tree-like structure, which we refer as label tree. The tree uses labels to rank the nodes. As a quality of measure we use the agony score.

We show that our problem is **NP**-hard, and inapproximable even if we limit the number of leaves in the label tree. This is in contrast with the original optimization problem: a ranking minimizing agony can be discovered in polynomial time.

Due to the **NP**-hardness of our problem, we resort to heuristics and propose a divide-and-conquer heuristic algorithm in Section 3 that runs in $\mathcal{O}((n+m)\log n + \ell R)$ time, where $R = \sum_v |L(v)|$ is the number of node-label pairs in the given graph, ℓ is the number of nodes in the resulting label tree, and n and m denote the number of nodes and edges respectively.

The algorithm starts with an empty tree and finds the best split minimizing the score. This process is continued until we have reached the maximum number of nodes or the score can no longer be increased. It is important to point out that the bottleneck here is computing the score while looking for the optimal split as a naive implementation would require to enumerate over all edges for every possible candidate. We avoid unnecessary enumerating over the edges by repurposing some results from Tatti [19] and maintaining certain counters. Such bookkeeping allows a fast, practical method to find label trees with low agony score.

Furthermore, we show experimentally in Section 5 that the algorithm performs well in practice, finds the hidden hierarchical structure in synthetic data, and finds explainable hierarchies in real-world datasets. Moreover, our algorithm is reasonably fast in practice as we are able to process large networks with over hundreds of thousands of edges in less than 30 minutes.

*This research is supported by the Academy of Finland projects MALSOME (343045)

†HIIT, Helsinki University, firstname.lastname@helsinki.fi

2 Preliminary notation and problem definition

The main input to our problem is a *weighted, directed, and labeled graph* which we denote by $G = (V, E, w, L)$, where V is a set of nodes, E is a set of edges, w is a function mapping an edge to a real positive number. If w is not provided, we assume that each edge has a weight of 1. L is a *label* function which assigns a set of labels from U to each vertex, where U is the label universe. Note that, each node can have multiple labels. We often denote $n = |V|$ and $m = |E|$. Given a subset of nodes W, we will write $E(W)$ to be the edges having both endpoints in W. Given two subsets of nodes W_1 and W_2, we write $E(W_1, W_2)$ to be the edges that go from W_1 to W_2.

Given $G = (V, E, w, L)$, our goal is to rank a set of vertices V. We express this ranking with a *rank assignment* r; a function mapping a vertex to an integer.

Given a graph G and a rank assignment r, we say that an edge (u, v) is *forward* if $r(u) < r(v)$, otherwise edge is *backward*, even if $r(u) = r(v)$. Ideally, rank assignment r should not have backward edges, meaning that, for any $(u, v) \in E$ we should have $r(u) < r(v)$.

Next, we define the penalty function p which penalizes the backward edges based on the severity of the violation. The penalty for a single edge (u, v) is equal to $p(d) = \max(0, d + 1)$, where $d = r(u) - r(v)$. The forward edges will always receive 0 penalty. Whenever $d \geq 0$, then the backward edges will receive $d+1$ amount of penalty. For example, an edge (u, v) with $r(u) = r(v)$ has a penalty score of 1. if $r(u) = r(v) + 1$, then the penalty becomes 2, and so on.

Next, we define a score for the ranking, to be the sum of all backward edge penalties multiplied by its weight as follows.

DEFINITION 2.1. *Assume a weighted directed graph $G = (V, E, w)$, a rank assignment r, and a penalty function p. We define* agony *score $q(G, r)$ as*

$$q(G, r) = \sum_{e=(u,v)\in E} w(e)p(r(u) - r(v)) \quad .$$

Finding rank assignment minimizing agony can be done in polynomial time [8, 19].

As mentioned earlier we are interested in explainable rankings. Let us now formalize this notion.

DEFINITION 2.2. *We define a* label tree, *T as follows. Label tree is an ordered, binary tree, where each leaf node represents a rank, which is an index starting from the leftmost leaf to the rightmost leaf. Each of the non-leaf node has a label t and a Boolean value c.*

We use the label tree T to partition the vertices of G by traversing them from the root to a leaf. Let α be a non-leaf node in T with a label t and a boolean value c. If c is true and, then a node v with $t \in L(v)$ traverses to the left branch, otherwise to the right. If c is false, the branch roles are reversed.

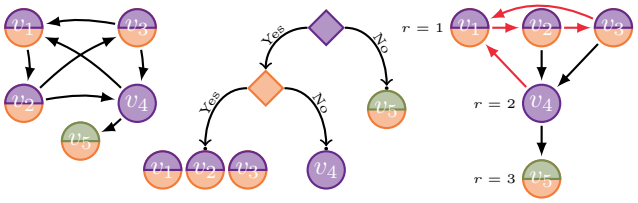

Figure 1: Graph G, label tree T, and the resulting hierarchy. The agony $q(G, T) = 5$.

The following notations will be useful in the next section. Given a graph G, a label tree T, and a node α in T, we write $V(\alpha)$ to be the nodes in G that traverse through or end up in α. We also write $E(\alpha) = E(V(\alpha))$. Given a label t, we write $V(\alpha, t)$ to be the nodes in $V(\alpha)$ that have label t. Finally, given a label t and Boolean value c, we write $V(\alpha, t, c)$ to be $V(\alpha, t)$ if c is true, and $V(\alpha) \setminus V(\alpha, t)$ if c is false.

Next, we define the score for the label tree.

DEFINITION 2.3. *Let $r(T, X)$ denote the index of the leaf after traversing the label tree T using X, where X is the label set. We define the* agony score *$q(G, T)$ as*

$$q(G, T) = \sum_{e=(u,v)\in E} w(e)p(r(T, L(u)) - r(T, L(v))) \quad .$$

Finally, we state our main optimization problem.

PROBLEM 2.1. (L-AGONY) *Given a weighted, directed, labeled graph $G = (V, E, w, L)$ and an integer k, find a label tree T, with at most k number of leaves, which minimizes the label agony $q(G, T)$.*

An example of a graph and a tree is given in Fig. 1.

We should point out that the cardinality constraint k is an optional parameter, the problem makes sense even if we set $k = \infty$.

3 Algorithm for finding tree

In this section we first show that our problem is **NP**-hard and present our greedy method for finding trees.

3.1 Computational complexity of L-agony

Finding rank r minimizing $q(G, r)$ can be done in polynomial time. However, our problem, that is, finding label tree minimizing $q(G, T)$ turns out to be **NP**-hard.

PROPOSITION 3.1. L-AGONY *is **NP**-hard.*

See Appendix (in the extended version of the paper) for all the proofs.

The proof of Proposition 3.1 shows that deciding whether there is a tree T yielding $q(G,T) = 0$ is **NP**-complete. This immediately implies that there is no approximation algorithm for L-AGONY since any approximation algorithm should be able to find optimal tree T if there is a solution yielding $q(G,T) = 0$.

COROLLARY 3.1. L-AGONY *is inapproximable, unless* $\boldsymbol{P = NP}$.

The proof of Proposition 3.1 relied that we set cardinality constraint k. It turns out that the problem is also **NP**-hard even if the constraint is not enforced.

PROPOSITION 3.2. L-AGONY *is* ***NP****-hard, even with* $k = \infty$.

3.2 Greedy algorithm Since we cannot solve L-AGONY efficiently or even approximate the constrained version of our problem, we need to resort to heuristics.

We approach the problem with a greedy method: starting from an empty tree, we find the best possible split, perform the split if there is a gain in score, and recurse on both leaves. The pseudo-code for the algorithm is given in Algorithm 1.

Here, we ignore any cardinality constraint. We will address the constraint in Section 3.5.

GREEDY uses two subroutines: TEST to compute the gain of split candidate, and CONSTRUCT to update the tree. In the next two sections, we discuss how these two subroutines can be implemented efficiently.

Algorithm 1: GREEDY(α). Calls TEST to find the best split. Calls CONSTRUCT to update the structures. Recurses to GREEDY for further splits.

```
find optimal label t and criterion c using TEST;
Δ ← reduction in score when splitting with (t, c);
if Δ < 0 then
    β, γ ← CONSTRUCT(α, t, c);
    Δ(α) ← Δ;
    GREEDY(β); GREEDY(γ);
```

3.3 Computing gain The computational bottleneck of the greedy algorithm is finding the next split: Given a rank function, computing agony from scratch requires $\mathcal{O}(m)$ time. Moreover, this calculation needs to be done for every viable label.

Luckily, we can speed this process by adopting the ideas presented by Tatti [19], where the authors propose a divide-and-conquer algorithm for finding ranks with low agony $q(G,r)$. Since the original algorithm does not use any label information, we will modify the approach so that it can be used for our problem.

More formally, we will maintain several counters that allow us to compute agony without enumerating over the edges.

Let T be a current tree and let α be a leaf in T with a rank i. Let U be the vertices with smaller rank, that is,

$$U = \{v \in U \mid r(T, L(v)) < i\} \quad .$$

Similarly, let W be the vertices with the larger rank

$$W = \{v \in U \mid r(T, L(v)) > i\} \quad .$$

We maintain 3 counters for each leaf and each vertex: Firstly, we maintain the total weight of backward edges that go over α,

$$b(\alpha) = \sum_{e \in E(W,U)} w(e), \tag{3.1}$$

secondly we maintain the total weight of backward edges from higher ranks to $V(\alpha)$,

$$ib(\alpha) = \sum_{v \in V(\alpha)} ib(v)\,, \quad \text{where} \quad ib(v) = \sum_{e \in E(W,v)} w(e), \tag{3.2}$$

and we maintain the total weight of backward edges from $V(\alpha)$ to lower ranks,

$$ob(\alpha) = \sum_{v \in V(\alpha)} ob(v)\,, \quad \text{where} \quad ob(v) = \sum_{e \in E(v,U)} w(e)\ . \tag{3.3}$$

Finally, for each vertex v in α we maintain the total weight of backward incoming edges minus the total weight of backward outgoing edges,

$$d(v) = \sum_{e \in E(W \cup V(\alpha), v)} w(e) - \sum_{e \in E(v, U \cup V(\alpha))} w(e)\ . \tag{3.4}$$

We will use the following result to compute the gain.

PROPOSITION 3.3. (PROPOSITION 8 IN [19]) *Let α be a leaf of a tree T. Assume a new tree T', where we have split α to two leaves. Let Y_1 be the vertex set of the new left leaf, and Y_2 the vertex set of the new right leaf. Then the score difference is*

$$q(T') - q(T) = b(\alpha) + ib(\alpha) - \sum_{y \in Y_2} d(y) \tag{3.5}$$

that can be rewritten as

$$q(T') - q(T) = b(\alpha) + ob(\alpha) + \sum_{y \in Y_1} d(y) \quad . \tag{3.6}$$

We make two observations: (1) We do not need to enumerate over edges. (2) We can test both cases for split (t, c) using only $V(\alpha, t)$: if $c = \mathit{true}$, then we set $Y_1 = V(\alpha, t)$ and use Eq. 3.6, if $c = \mathit{false}$, then we set $Y_2 = V(\alpha, t)$ and use Eq. 3.5. The pseudo-code for this procedure is given in Algorithm 2.

Algorithm 2: TEST(α, t), computes the gain difference of α due to split based on label t and criterion c. Tests both criteria $c = \mathit{true}$ and $c = \mathit{false}$ and returns the better.

$\Delta_1 \leftarrow b(\alpha) + ob(\alpha) + \sum_{y \in V(\alpha,t)} d(y; \alpha)$;
$\Delta_2 \leftarrow b(\alpha) + ib(\alpha) - \sum_{y \in V(\alpha,t)} d(y; \alpha)$;
return $\min(\Delta_1, \Delta_2)$, $\Delta_1 \leq \Delta_2$;

3.4 Maintaining the tree Once we have found the best candidate for splitting α, we need to split the leaf α into new leaves β, and γ.

More specifically, we need to update the counters $b(\cdot)$, $ib(\cdot)$, $ob(\cdot)$, and $d(\cdot)$, as well as compute $V(\beta)$ and $V(\gamma)$. Here, we modify the algorithm in [19] to suit our needs.

Computing the vertex sets $V(\beta)$ and $V(\gamma)$ can be done in $\mathcal{O}(|V(\alpha, t)|)$ time by first moving $V(\alpha)$ to one of the child, say γ, in constant time, and then moving the extra nodes in $\mathcal{O}(|V(\alpha, t)|)$ time.

In order to update the counters we will need to iterate over the cross-edges between β and γ. We can do this iteration either by going over the edges adjacent to $V(\beta)$, or by going over the edges adjacent to $V(\gamma)$.

Both approaches will yield the same result, so we use the one that visits fewer unnecessary edges. More specifically: if $|V(\beta)| + |E(\beta)| \leq |V(\gamma)| + |E(\gamma)|$, enumerate over $V(\beta)$, otherwise we enumerate over $V(\gamma)$.

The pseudo-code for CONSTRUCT is given in Algorithm 3. A straightforward calculation, which we will omit, starting from Eqs. 3.1–3.4 demonstrates that Algorithm 3 maintains the counters correctly. Moreover, since we need the cross-edges between β and γ only once, we will delete them as we are processing them.

Next we prove the computational complexity of GREEDY.

PROPOSITION 3.4. *The running time of* GREEDY *is in* $\mathcal{O}((n+m)\log n + \ell R)$, *where* $R = \sum_v |L(v)|$ *is the number of node-label pairs in* G, *and* ℓ *is the number of nodes in the resulting label tree.*

3.5 Applying cardinality constraint by pruning As our last step, we look on how to enforce possible

Algorithm 3: CONSTRUCT(α), splits α

$N \leftarrow V(\alpha, t, c)$;
$P \leftarrow V(\alpha) \setminus N$;
create a new leaf β with $V(\beta) = N$;
create a new leaf γ with $V(\gamma) = P$;
if $|V(\beta)| + |E(\beta)| \leq |V(\gamma)| + |E(\gamma)|$ **then**
  $ib(\beta) \leftarrow \sum_{x \in N} ib(x)$; $ib(\gamma) \leftarrow ib(\alpha) - ib(\beta)$;
  $ob(\beta) \leftarrow \sum_{x \in N} ob(x)$; $ob(\gamma) \leftarrow ob(\alpha) - ob(\beta)$;
  $b(\beta) \leftarrow b(\alpha) + ob(\gamma)$; $b(\gamma) \leftarrow b(\alpha) + ib(\beta)$;
  foreach $x \in N$ **do**
    foreach $e = (z, x)$ *such that* $z \in P$ **do**
      add $w(e)$ to $ib(x)$, $ib(\beta)$, $ob(z)$, $ob(\gamma)$;
      delete e;
    foreach $e = (x, z)$ *such that* $z \in P$ **do**
      decrease $d(x)$, $d(z)$ by $w(e)$;
      delete e;
else
  $ib(\gamma) \leftarrow \sum_{x \in P} ib(x)$; $ib(\beta) \leftarrow ib(\alpha) - ib(\gamma)$;
  $ob(\gamma) \leftarrow \sum_{x \in P} ob(x)$; $ob(\beta) \leftarrow ob(\alpha) - ob(\gamma)$;
  $b(\beta) \leftarrow b(\alpha) + ob(\gamma)$; $b(\gamma) \leftarrow b(\alpha) + ib(\beta)$;
  foreach $z \in P$ **do**
    foreach $e = (z, x)$ *such that* $x \in N$ **do**
      add $w(e)$ to $ib(x)$, $ib(\beta)$, $ob(z)$, $ob(\gamma)$;
      delete e;
    foreach $e = (x, z)$ *such that* $x \in N$ **do**
      decrease $d(x)$, $d(z)$ by $w(e)$;
      delete e;
return β, γ ;

cardinality constraint. Given the cardinality constraint k and a label tree T produced by GREEDY, we can reduce the number of leaves by pruning some branches of T. The optimal subtree can be obtained using dynamic programming, as proposed by Tatti [19].

Let us define $opt(\alpha; h)$ be the optimal gain that obtained in the branch in T starting from the node α by using only h leaves (and pruning the remaining nodes).

If α is the root of T, then $opt(\alpha; k)$ is the optimal agony achieved by pruning T to have only k leaves. We initialize the array by setting $opt(\alpha; 1) = 0$ for any α, and $opt(\alpha; h) = 0$ if α is a leaf in T. If α is a non-leaf and $k > 1$, then we use the identity.

$$opt(\alpha; h) = \Delta(\alpha) + \min_{1 \leq \ell \leq h-1} opt(\beta; \ell) + opt(\gamma; h - \ell)\,,$$

where β is the left child and γ is the right child, and $\Delta(\alpha)$ is the improvement in agony as recorded in GREEDY. To perform the trace-back, we further record the optimal index ℓ in a different table. Computing

$opt(\alpha; h)$ requires $\mathcal{O}(k)$ time. We compute at most $\mathcal{O}(nk)$ entries which leads to $\mathcal{O}\big(nk^2\big)$ running time.

4 Related work

The notion of agony score for ranking was first proposed by Gupte et al. [8] for the unweighted, directed graphs. The authors provided a polynomial-time, exact algorithm in which agony minimization problem has been formulated as an integer linear program. They exploits primal-dual techniques where the primal problem tries to minimize the agony and the dual finds the maximum eulerian subgraph. Later, Tatti extended agony minimization problem to handle weights and the cardinality constraint [19]. At the same time, the author introduced a greedy heuristic to find hierarchies provably faster. We should stress that these algorithms do not use any label information whereas we propose an optimization problem and a heuristic algorithm in which the node labels are exploited.

The appeal of using agony is that it can be solved in polynomial time. Alternatively, if we use constant penalty function for backward edges, the minimization problem is equivalent to FEEDBACK ARC SET (FAS), which is known to be **APX**-hard with a coefficient of $c = 1.3606$ [3], no known constant-ratio approximation algorithm with the best known approximation algorithm yielding a ratio $\mathcal{O}(\log n \log\log n)$ [5]. Interestingly, while Tatti [19] showed minimizing agony for a convex penalty, that is, $q(G, r)$ can be done in polynomial time; however, in this paper, we show that finding label tree minimizing $q(G, T)$ turns out to be **NP**-hard.

Let us now look at alternative methods for ranking nodes in a graph. We should stress that these methods do not use any label information.

A popular and a classic method for ranking players in competitions, is Elo ranking, originally proposed as a rating method to rank chess players, based on the outcomes of tournament games [4].

Maiya and Berger-Wolf [12] inferred rankings, by first inducing interaction model to the network and then searching maximum likelihood hierarchy for each candidate model, with a greedy heuristic.

Another way of finding hierarchies is to use graph theoretic centrality measures. The notion of eigenvector centrality score and degree has been proposed by Memon et al. [13], assuming that high scores imply highly ranked entities. Utilizing a weighted combination of such graph theoretic measures, including number of cliques and betweenness centrality was proposed by Rowe et al. [17]. Highly ranked nodes in directed graphs typically have many outgoing edges. Finding such nodes can be also done using a classic HITS method introduced by Kleinberg [10], where the score of a hub is based on the quality of the nodes it points.

5 Experimental evaluation

In this section we present our experimental evaluation which has the following goals: (*i*) Test how well GREEDY finds ground-truth hierarchy in synthetic dataset, and assess the impact of prevailing noise level in labels, in terms of the Kendall's τ coefficient.[1] (*ii*) Compare the agony scores of GREEDY against the scores of hierarchies obtained without using any labels. Here we used [19], which we call AGONY. (*iii*) Study how the constraint k influences the agony score. (*iv*) Explore the hierarchies in real-world labeled datasets.

We implemented the algorithm in Python[2] and used a 2.4GHz Intel Core i5 processor and 16GB RAM.

Synthetic datasets: To test our algorithm, we generated 7 synthetic networks in which the characteristics are stated in Table 1. For each network, we first set our basic parameters; number of nodes n, number of edges m, and number of ranks h. Next we randomly assigned the ranks for each node, with equal probability. Note that, each rank has its own specific label, we call them as *true* labels. Initially, we assigned true labels to vertices such that a node which belongs to the rank r_i, receives the label l_i. This can be considered as true label assignment which is done prior to the noisy label assignment. Afterwards, we fixed two noise parameters, θ and μ, where θ indicates the percentage of nodes which contains *false* labels and μ indicates the percentage of nodes which contains *noise* labels. To add those *false* labels, we first chose a subset of vertices in accordance with θ and then added a random label chosen from label universe U_{true} to each node, without repeating the available labels for that node. Similarly, we assigned *noise* labels for a percentage of nodes determined by parameter μ, randomly chosen from U_{noise}, a label set of size 100.

After being set all noise parameter values, we fixed a certain percentage value, η as a controlling parameter of the proportion of forward edges. Next, we uniformly sampled m number of vertex pairs as follows: For each rank $i = 1, \ldots, h-1$, we generated $m/(h-1)$ edges, between the nodes with rank i and rank $i+1$. For each vertex pair, we determined whether the chosen edge is either forward or backward. This choice can be viewed in terms of a coin-flipping experiment of a biased coin with probability η. Finally, we removed all self-loops and aggregated the same directed edges together by assigning their accumulated weights.

[1] A measure of rank correlation which indicates the similarity between two rankings.

[2] See `https://version.helsinki.fi/dacs/` for the code.

Table 1: Characteristics of the synthetic datasets. Here, h indicates the number of hierarchy levels (ranks), η is the percentage of forward edges, θ and μ implies the percentage of vertices which contains false labels and noise labels respectively.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	h	θ	μ	η
Syn-1	4 000	3 200	5	0.04	0.05	0.6
Syn-2	5 000	7 000	8	0.04	0.06	0.7
Syn-3	1 200	1 200	5	0.05	0.07	0.65
Syn-4	1 000	1 750	6	0.05	0.08	0.9
Syn-5	4 000	4 200	7	0.06	0.09	0.85
Syn-6	1 000	5 600	15	0.08	0.1	0.8
Syn-7	40 000	135 000	10	0.08	0.11	0.75

Table 2: Statistics from the experiments with the synthetic datasets. Here, q_{true} and q_{dis} are the ground truth and discovered agony scores respectively, q_{base} is the discovered agony using AGONY, h_{dis} and h_{base} are the discovered number of ranks using GREEDY and AGONY, k_{tau} is the Kendall's τ coefficient, and *time* gives the computational time in seconds for GREEDY.

Dataset	q_{true}	q_{dis}	q_{base}	h_{dis}	h_{base}	$k\tau_{dis}$	$k\tau_{base}$	*time*
Syn-1	2 242	2 322	10	5	25	0.97	0.1	2.78
Syn-2	4 160	4 453	648	8	47	0.97	0.29	6.47
Syn-3	782	798	2	5	20	0.97	0.15	0.5
Syn-4	388	535	2	7	27	0.96	0.58	0.62
Syn-5	1 268	1 591	16	8	22	0.96	0.26	4.21
Syn-6	2 262	3 237	2 158	15	19	0.95	0.96	1.29
Syn-7	66 978	89 017	51 504	10	24	0.93	0.79	189.95

Real-world datasets: We explore label hierarchies in 7 publicly available, labeled, real-world datasets. The details of the datasets are shown in Table 3. *EIES* is a network of researchers working on social network analysis, known as *Freeman's EIES networks*[3]. It contains a set of message links among 32 researchers, weighted based on the number of messages sent. Each researcher is attributed based on his/her main disciplinary affiliation and the number of citations each researcher had in the social science citation index in 1978. *School* dataset[4] contains directed contact links between students in a high school in France, weighted by the length of interval where the contact was active. Students are labeled with their class labels. *Cora* [16][5] and *Cite-seer* [16][5] datasets are citation networks where papers are attributed with a category and a class respectively. *Patent-citation* [11][6] contains patent citations data. Each patent is a vertex and each citation is an edge in our network. Each patent has a class, a technological category, and a technological sub-category which we consider as node labels. For our experiments, we extracted a sub-network which contains first 7 000 citations and then filtered out the nodes whose category details are missing from the dataset. *DBLP-citation* [18][7] contains citation relationships from top venues in Data Mining and Machine Learning (SDM, NIPS, ICDM, KDD, ECMLPKDD, and WWW). We extracted set of labels by stemming the titles of the papers and removing stop words. *Physics-citation*[8] is a citation network extracted from Journals in the High Energy Physics Theory category. Herein, we use the title of the publication as labels, after stemming the titles and removing stop words.

Results of synthetic datasets: The detailed statistics and the running times are given in Table 2. First, we observe $2nd$, $3rd$, and $4th$ columns in Table 2. These columns display ground truth agony scores of the datasets, followed by the scores obtained with GREEDY and AGONY. Generally, the discovered agony score obtained by GREEDY is more closer to the ground truth agony score, as opposed to the score found by AGONY. Secondly, consider $k\tau_{dis}$ column of Table 2, which exhibits the Kendall's τ measures attained by our algorithm. For all these experiments Kendall's τ coefficient exceeds 0.9; which indicates a high quality match of the discovered hierarchies with compared to the ground truth, even under the presence of 10% false labels.

Let us now compare Kendall's τ measures achieved by GREEDY with the scores by AGONY, which is shown in the $k\tau_{base}$ column. We can observe that Kendall's τ measures obtained by GREEDY is significantly higher than the ones by baseline, except for one outlier case. Next, let us consider the $5th$ column of Table 2, which shows the number of ranks discovered by GREEDY. Interestingly, this is equal to the number of ground truth ranks. Moreover, the numbers of ranks discovered by AGONY ($6th$ column) are much higher than the ground truth. Finally, from the computational time results which are stated in the last column in Table 2, we see that the running times are reasonably practical to run our algorithm nearly in 3 minutes for a graph with over 40 000 vertices and 100 000 edges.

[3] https://toreopsahl.com/datasets/
[4] http://www.sociopatterns.org/datasets/
[5] https://networkrepository.com
[6] https://www2.helsinki.fi/en/researchgroups/unified-database-management-systems-udbms/datasets/patent-dataset
[7] https://www.aminer.org/citation
[8] https://github.com/chriskal96/physics-theory-citation-network

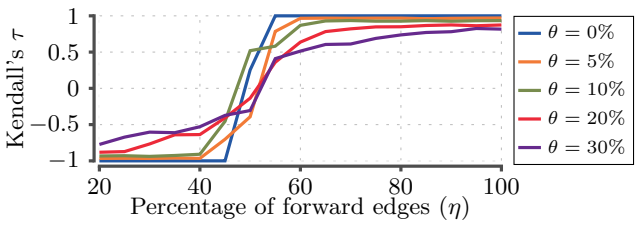

Figure 2: Kendall's τ coefficient as a function of η for the cases of several false label probabilities (θ) shown in the legend. This experiment is done for $|V| = 4\,000$, $|E| = 7\,000$, $\mu = 0.05$, and $h = 10$ using GREEDY.

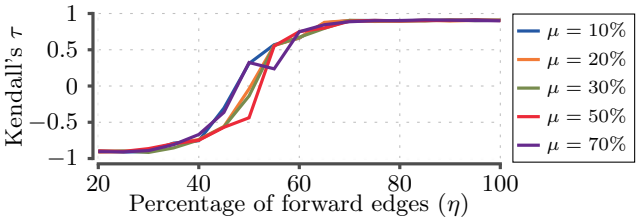

Figure 3: Kendall's τ coefficient as a function of η for the cases of several noise label probabilities (μ) shown in the legend. This experiment is done for $|V| = 4\,000$, $|E| = 7\,000$, $\theta = 0.15$, and $h = 10$ using GREEDY.

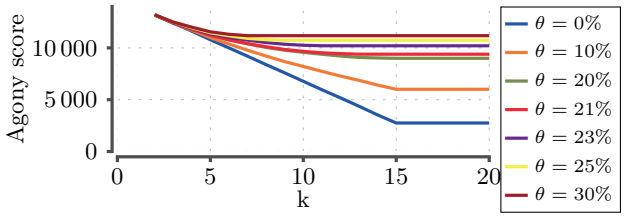

Figure 4: Agony score ($q(T)$) as a function of the constraint k for the cases of several false label probabilities (θ) as shown in the legend. This experiment is done for $|V| = 4\,000$, $|E| \approx 7\,000$, $h = 15$, $\mu = 0.05$, and $\eta = 0.9$ using GREEDY.

Table 3: Characteristics of the real-world datasets. Here, $|T|$ is the number of labels.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$
EIES	32	460	9
School	329	502	9
Cora	2 708	5 429	7
Cite-seer	3 264	4 536	6
Patent-citation	5 439	4 232	394
DBLP-citation	30 581	70 972	10 245
Physics-citation	29 555	352 807	4 715

Let us now consider Figure 2 which demonstrates how Kendall's τ metric changes with respect to the percentage of forward edges in the network. In general, we see that Kendall's τ gradually increases when the percentage of forward edges increases. The coefficient significantly increases nearly at the 50% percentage, from negative to positive, due to the fact that there are more forward edges than backward edges after 50%. We repeat this experiment by adjusting the false label probability θ to be $0\%, 5\%, 10\%$, 20%, and 30%. When there are no false labels, the coefficient achieves 1 which implies a perfect match in hierarchy, while the percentage of forward edges reaches nearly 50%. The coefficient achieves its max at 60% for the case of 5% of false labels, whereas same is obtained nearly at 95% for the case of 30% of false labels. Hence we conclude that lesser the amount of *false* label noise is present, more accurate hierarchies could be extracted and more percentage of backward edges can be withstood.

Let us now investigate the effect of noise label probability μ; which is shown in Figure 3. On contrary to the previous experiment, we fix θ and then repeat the same experiment by varying μ. Unlike false label probability θ, even if we increase μ to be 70%, we can conclude that μ does not significantly affect our performance metric, Kendall's τ coefficient.

Next Figure 4 demonstrates the agony score as a function of k. Here we repeat the experiment at different false label probabilities such as $0\%, 10\%, 20\%, 21\%, 23\%, 25\%$, and 30%. From the results we see that, for the first 3 cases, in which no false labels, 10% and 20% of false labels, agony score decreases more prominently until it approaches $k = 15$ and then it remains constant even if we increase k further. Similar phenomenon happens at $k = 14$, $k = 13$, $k = 9$, and $k = 8$, for the cases of $21\%, 23\%, 25\%$, and 30% of false labels respectively. This is because in latter experiments the tree contains at most 14, 13, 9, and 8 leaves and can not improve the agony score by splitting further. In addition, we observe that the elbow point is reached at the earliest for *high-noisy* labels, with compared to the *low-noisy* labels.

Results of real-world datasets: For each dataset, we computed the agony using GREEDY and AGONY. The statistics of discovered hierarchical structures for real-world datasets are shown in Table 4. First, let us look at the discovered number of ranks, which are given in Table 4. Generally, h_{base} is higher with compared to h_{dis} except for *EIES* and *Patent-citation* datasets. Next, *2nd* and *3rd* columns in Table 4 dis-

Table 4: Experimental details with real-world datasets. Here, q_{dis} and q_{base} are discovered agony using GREEDY and AGONY respectively, Similarly, h_{dis} and h_{base} indicate the number of ranks discovered, d gives the height of the tree constructed using GREEDY, and *time* gives the computational time for GREEDY.

Dataset	q_{dis}	q_{base}	h_{dis}	h_{base}	d	*time*
EIES	14 035	12 682	4	3	4	3.4ms
School	1 426	982	3	10	3	20ms
Cora	5 351	340	3	18	3	35s
Cite-seer	4 397	0	3	16	3	1.3s
Patent-citation	4 074	0	7	3	4	4m
DBLP-citation	68 511	329	10	38	6	43m
Physics-citation	339 475	2 268	9	236	5	22m

Table 5: Details of two label-free networks constructed using *Cora* and *Cite-seer* datasets. Here, q_{dis} is the discovered agony using GREEDY, q_{new} gives discovered agony using AGONY, on newly formed label-free network, h indicates the discovered number of ranks using GREEDY, and h_{new} indicates the discovered number of ranks using AGONY on label-free network.

Dataset	$\lvert V_{new}\rvert$	$\lvert E_{new}\rvert$	q_{dis}	q_{new}	h	h_{new}
Cora	7	39	5 351	5 315	3	3
Cite-seer	6	30	4 397	4 397	3	3

play agony scores obtained by GREEDY and AGONY. As you can see in q_{base}, $q(G, r) = 0$ is reached twice for *Cite-seer* and *Patent-citation* datasets by AGONY. The column d in Table 4 shows the depth of the constructed tree which is significantly smaller than $|V|$. The last column in Table 4 shows that running times are reasonably practical so that we could construct the label tree for a label graph with more than 300 000 edges, 80 000 vertices, and 4 000 labels in less than 22 minutes.

Note that we can solve L-AGONY exactly if each node has only a single label with a new graph by using the set of labels as new nodes and combined edges as its edges. We then run the AGONY algorithm for this new label-free network to obtain the optimal rank. We do this for *Cora* and *Cite-seer* datasets as each paper has only one label. The results of those experiments are shown in Table 5. Let us focus on q_{dis} and q_{new} columns in Table 5. The discovered agony scores obtained using our algorithm and baseline are approximately similar, on newly formed label-free networks, implying that our heuristic finds either optimal or close to optimal rankings for these networks.

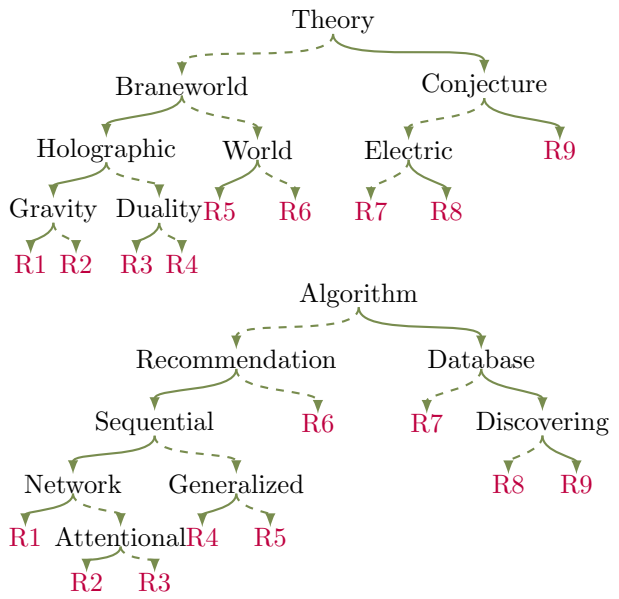

Figure 5: Discovered label trees for *Physics* (above) and *DBLP* (below) datasets. Solid lines indicate the branch for the nodes with the corresponding label.

Case study: To conclude the experimental section, we take a closer look at hierarchical groups found by GREEDY for some of the datasets. We highlight two datasets, additional trees are given in Appendix.

The label tree obtained for *Physics-citation* dataset is given in the top of Figure 5. Our algorithm first partitions the physics publications based on whether it is theoretical or application-based, suggesting that application-based papers cite more theoretical papers. Next, the label *conjecture* divides the set of theoretical papers further in to two halves, suggesting that more conjectures are likely to be presented in theoretical papers. Application-based publications are then divided further with the label *braneworld* which is a cosmology related scientific term. The labels like *holographic*, *gravity*, and *braneworld* are in the same line of research which is sensible.

The label tree obtained for *DBLP-citation* dataset is shown in the bottom tree of Figure 5. This tree shows 9 ranks in total. Our algorithm first partitions the publications based on whether the title contains the word, *algorithm*, suggesting that the papers which contain word, *algorithm* are cited more by the papers which does not contain the word, *algorithm*. Likewise, this label tree postulates that the publications which contain the words, *algorithm*, *database*, and *discovering* are cited more by the publications which contain *recommendation*, *sequential*, and *network* in their titles.

6 Concluding remarks

We introduced a novel tree-based, hierarchy mining problem for vertex-labeled, directed graphs. Here the goal is to rank the nodes into tiers so that ideally the edges are directing to the lower ranks. The ranking is done with a decision tree that can only use node labels.

The goal was to minimize a penalty score known as agony which penalized the edges from higher ranks to lower ranks. We showed that the construction of such a label tree optimally is **NP**-hard, or even inapproximable when we limit the number of leaves. Therefore, we presented a heuristic algorithm which runs in $\mathcal{O}((n+m)\log n + \ell R)$, where $R = \sum_v |L(v)|$ is the number of node-label pairs in given graph, ℓ is the number of nodes in the resulting label tree, and n and m are the number of nodes and edges respectively. To enforce the cardinality constraint for number of ranks k, we pruned the label tree such that tree had only k leaves, exploiting dynamic programming techniques.

The synthetic experiments showed that our approach accurately recovers the latent hierarchy. The experiments on real-world networks confirmed that our proposed label-driven algorithm achieved a good quality ranks which can be explained by node labels. We discovered hierarchies reasonably fast in practice. The notion of discovering hierarchies in labeled networks opens up several lines of work. For example, instead of requiring that the nodes in each rank group match exactly to the label tree we can require that only a large portion of the nodes match the label tree.

References

[1] Z. Bai, S. Ravi, and I. Davidson. Towards description of block model on graph. In *ECMLPKDD*, pages 37–53, 2020.

[2] C. Bothorel, J. D. Cruz, M. Magnani, and B. Micenkova. Clustering attributed graphs: models, measures and methods. *Network Science*, 3(3):408–444, 2015.

[3] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[4] A. E. Elo. *The rating of chessplayers, past and present*. Arco Pub., New York, 1978.

[5] G. Even, B. Schieber, M. Sudan, et al. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[6] I. Falih, N. Grozavu, R. Kanawati, and Y. Bennani. Community detection in attributed network. In *WWW*, pages 1299–1306, 2018.

[7] E. Galbrun, A. Gionis, and N. Tatti. Overlapping community detection in labeled graphs. *DMKD*, 28 (5):1586–1610, 2014.

[8] M. Gupte, P. Shankar, J. Li, S. Muthukrishnan, and L. Iftode. Finding hierarchy in directed online social networks. In *WWW*, pages 557–566, 2011.

[9] K. A. Jameson, M. C. Appleby, and L. C. Freeman. Finding an appropriate order for a hierarchy based on probabilistic dominance. *Animal behaviour*, 57 (5):991–998, 1999.

[10] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM*, 46(5):604–632, 1999.

[11] J. Lu, J. Chen, and C. Zhang. Helsinki Multi-Model Data Repository. `https://www.helsinki.fi/en/researchgroups/unified-database-management-systems-udbms/datasets/patent-dataset`, 2018.

[12] A. S. Maiya and T. Y. Berger-Wolf. Inferring the maximum likelihood hierarchy in social networks. In *CSE*, volume 4, pages 245–250, 2009.

[13] N. Memon, H. L. Larsen, D. L. Hicks, and N. Harkiolakis. Retracted: detecting hidden hierarchy in terrorist networks: some case studies. In *ISI*, pages 477–489, 2008.

[14] S. Neumann, J. Ritter, and K. Budhathoki. Ranking the teams in european football leagues with agony. In *MLSA@ECMLPKDD*, 2018.

[15] S. Pool, F. Bonchi, and M. v. Leeuwen. Description-driven community detection. *TIST*, 5 (2):1–28, 2014.

[16] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL `https://networkrepository.com`.

[17] R. Rowe, G. Creamer, S. Hershkop, and S. J. Stolfo. Automated social hierarchy detection through email network analysis. In *WebKDD/SNA-KDD*, pages 109–117, 2007.

[18] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *KDD*, pages 990–998, 2008.

[19] N. Tatti. Tiers for peers: a practical algorithm for discovering hierarchy in weighted networks. *DMKD*, 31(3):702–738, 2017.

7 Proofs

Proof. [Proposition 3.1] We will prove the hardness from k-Cover, a problem where we are given a family of subsets $\mathcal{C}$ of a universe U and we asked if there are at most k sets covering U.

Assume that we are given an instance of Cover. Our graph is as follows: the vertices V consists of the

items in $U = u_1, \ldots, u_n$ plus one additional vertex w. We connect each u_i to w. The labels $L(u_i)$ are the indices of the sets in which u_i is included. Finally, $L(w) = \emptyset$.

We claim that there is a label tree T with at most $k+1$ leaves yielding $q(G,T) = 0$ if and only if there is a k-cover.

Assume there is a cover $\mathcal{Z}$. Then let T be a tree with $|\mathcal{Z}|$ non-leaves. Each of these nodes have a leaf as the left child, and one node has two leaves as children. Each non-leaf has a label corresponding to a set in $\mathcal{Z}$ guiding the nodes in G with the label to the left leaf. The tree has at most $k+1$ leaves, moreover the last leaf contains only w since it does not have any labels and $\mathcal{Z}$ is a cover. Consequently, all edges are forward, and $q(G,T) = 0$.

On the other hand, if $q(G,T) = 0$, then the last leaf contains only w since otherwise there is a backward edge. Let $\mathcal{Z}$ be the sets corresponding to the labels occurring in T. Then $\mathcal{Z}$ is a cover since otherwise there would be a node with the same rank as w, violating the assumption. Moreover, $|\mathcal{Z}| \leq k$ proving the claim. ∎

Proof. [Proposition 3.2] We will prove the hardness from k-Cover. Assume that we are given an instance of Cover: a universe U of n items, a family of m subsets $\mathcal{C}$, and an integer k.

Our graph is as follows: the vertices V consists of the items in $U = u_1, \ldots, u_m$, m vertices representing the sets $S = s_1, \ldots, s_m$, and one additional vertex x. We connect each u_i to x with a weight $w(e) = m + k(k+1)$. We connect x to each s_i with a weight $w(e) = 1$.

The labels $L(u_i)$ are the indices of the sets in which u_i is included. The single label for s_i is the index of the ith set. Finally, $L(w) = \emptyset$.

We claim that there is a label tree T yielding $q(G,T) \leq m + k(k+1)/2$ if and only if there is a k-cover.

Assume there is a cover $\mathcal{Z}$. Then let T be a tree with $|\mathcal{Z}|$ non-leaves. Each of these nodes have a leaf as the left child, and one node has two leaves as children. Each non-leaf has a label corresponding to a set in $\mathcal{Z}$ guiding the nodes in G with the label to the left leaf. Since $\mathcal{Z}$ is a cover all edges (u_i, x) are forward. Moreover, k vertices in S have ranks $1, \ldots, k$ and the remaining $m-k$ vertices in S have the same rank as x, that is $k+1$. Consequently,

$$q(G,T) = m + k(k+1)/2 \quad .$$

On the other hand, if $q(G,T) \leq m + k(k+1)/2$, then the last leaf contains only x and vertices from S, since $w(u_i, x) > m + k(k+1)/2$.

Let $\mathcal{Z}$ be the o sets corresponding to the labels occurring in the path to x from the root, in order. Then $\mathcal{Z}$ is a cover since otherwise there is a node in U with the same rank as x, violating the assumption. Let $z_1, \ldots, z_o$ be the corresponding vertices in S. Then $r(x) - r(z_i) \geq o - i + 1$ since there are at least $o - i$ leaves between z_i and x. Since x has the largest rank, we also have $r(x) \geq r(s_i)$. Consequently,

$$\begin{aligned} m + \frac{k(k+1)}{2} \geq q(G,T) &= \sum_{i=1}^{m} 1 + r(x) - r(s_i) \\ &\geq m - o + \sum_{i=1}^{o} (o - i + 2) = m + \frac{o(o+1)}{2}, \end{aligned}$$

which shows that $o \leq k$, proving the result. ∎

Proof. [Proposition 3.4] Greedy(α) first finds the best candidate by calling repeatedly Test(α, t), and then performs the split by calling Construct(α).

Since Test(α, t) runs in $\mathcal{O}(|V(\alpha,t)|)$ time, the total time needed to find the candidate for splitting α is $\mathcal{O}(\sum_t |V(\alpha,t)|) \subseteq \mathcal{O}(R)$.

Next we bound the time needed to update the counters during the split. Let us define $i_{e\alpha} = 1$ if an edge e is visited during Construct(α), and $i_{e\alpha} = 0$ otherwise. Define also $i_{v\alpha} = 1$ if a node v is visited during for-loop, and $i_{v\alpha} = 0$ otherwise.

Write $n_\alpha = \sum i_{v\alpha}$ and $m_\alpha = \sum i_{e\alpha}$ to be the total number of nodes and edges visited during the split.

Let us define $c_\alpha = |E(\alpha)| + |V(\alpha)|$. Testing whether $c_\beta \leq c_\gamma$ can be done in $\mathcal{O}(n_\alpha + m_\alpha)$, and splitting α can be done in $\mathcal{O}(|V(\alpha,t)| + n_\alpha + m_\alpha)$ time.

First note that $|V(\alpha,t)| \leq R$. The proposition then follows if we can prove that $\sum_\alpha n_\alpha + m_\alpha \in \mathcal{O}((n+m)\log n)$. Since

$$\sum_\alpha n_\alpha + m_\alpha = \sum_v \sum_\alpha i_{v\alpha} + \sum_e \sum_\alpha i_{e\alpha},$$

we will prove the claim by showing that $\sum_\alpha i_{e\alpha} \in \mathcal{O}(\log n)$ and $\sum_\alpha i_{v\alpha} \in \mathcal{O}(\log n)$

Fix edge e, and let α and α' be two nodes for which $i_{e\alpha} = i_{e\alpha'} = 1$. Then α' is a descendant of α, or otherwise. Assume the former. Then α' is either a child of α or is a descendant of that child. Let us denote this child by β and let γ be the other child.

Since $e \in E(\alpha') \subseteq E(\beta)$, we have $c_\beta \leq c_\gamma$ as otherwise e is not visited when α is split. Consequently,

$$2c_{\alpha'} \leq 2c_\beta \leq c_\beta + c_\gamma \leq c_\alpha \quad .$$

To conclude, let $\{\alpha_j\}$ be the nodes for which $i_{e\alpha_j} = 1$. We have shown that we can safely assume that α_{j+1} is a descendant of α_{j+1}. Moreover, $c_{\alpha_j} \geq 2c_{\alpha_{j+1}}$. Since

$c_{\alpha_1} \leq m + n$, there can be at most $\mathcal{O}(\log m) \subseteq \mathcal{O}(\log n)$ nodes. The argument for $\sum_\alpha i_{v\alpha} \in \mathcal{O}(\log n)$ is similar. ∎