Faster way to agony
Discovering hierarchies in directed graphs

Nikolaj Tatti

Helsinki Institute for Information Technology
Department of Information and Computer Science
Aalto University, Finland
nikolaj.tatti@aalto.fi

Abstract. Many real-world phenomena exhibit strong hierarchical structure. Consequently, in many real-world directed social networks vertices do not play equal role. Instead, vertices form a hierarchy such that the edges appear mainly from upper levels to lower levels. Discovering hierarchies from such graphs is a challenging problem that has gained attention. Formally, given a directed graph, we want to partition vertices into levels such that ideally there are only edges from upper levels to lower levels. From computational point of view, the ideal case is when the underlying directed graph is acyclic. In such case, we can partition the vertices into a hierarchy such that there are only edges from upper levels to lower edges. In practice, graphs are rarely acyclic, hence we need to penalize the edges that violate the hierarchy. One practical approach is agony, where each violating edge is penalized based on the severity of the violation. The fastest algorithm for computing agony requires $O(nm^2)$ time. In the paper we present an algorithm for computing agony that has better theoretical bound, namely $O(m^2)$. We also show that in practice the obtained bound is pessimistic and that we can use our algorithm to compute agony for large datasets. Moreover, our algorithm can be used as any-time algorithm.

Keywords: Graph mining, agony, hierarchy discovery, primal-dual, maximum eulerian subgraph

1 Introduction

Many real-world phenomena exhibit strong hierarchical structure [2, 5, 9, 10, 11]. For example, it is more likely that a manager in a large company will write emails to the her subordinates than an employee writes an email to his manager. As another example, in a tournament, it is more likely that a better team will win a second-tear team.

Discovering hierarchy in the context of directed networks can be viewed as the following optimization problem. Given a directed graph, partition vertices into levels such that there are only edges from upper levels to lower levels. For example, consider an email communication network of a large institute, directed edge $x \rightarrow y$ is created if x has written an email to y. We should expect that the upper level of the hierarchy consists of top-level managers and each level consists of subordinates of the previous level.

Unfortunately, such a partition is only possible when the graph does not have cycles, a rare case in practice. Instead a more fruitful approach is to find a hierarchy that minimizes some cost function. One possible cost function is to penalize every edge that violates the hierarchy with a constant cost. Unfortunately, this problem leads to Feedback Arc Set problem, where we are asked to discover a maximal directed acyclic subgraph. This problem is a classic **NP**-hard problem [4].

A practical variant of discovering hierarchies that was introduced recently by Gupte et al. [7] is to weight the edges based on the severity of the violation of hierarchy. Unlike the constant weights, this problem can be solved in $O(nm^2)$, polynomial time, where n is the number of vertices and m is the number of edges.

In this paper we introduce a new algorithm for computing a hierarchy that minimizes agony. Our algorithm achieves computational complexity of $O(m^2)$ which is significantly better than $O(nm^2)$, the computational complexity of the currently best approach. We also demonstrate empirically that $O(m^2)$ is in fact pessimistic and that we can compute agony using our approach for large networks.

Our approach is based on a primal-dual technique. Minimizing agony has an interpretable dual problem, finding eulerian subgraph, a graph where the in-degree is equal to the out-degree for each vertex, with the maximum number of edges. This relation implies that the agony will always be at least as large as any eulerian subgraph. We are able to exploit this relation by designing an iterative algorithm. At each iteration we decrease the gap between the current agony and the current eulerian subgraph by either modifying the hierarchy or modifying the eulerian subgraph. We show that each iteration requires only $O(m)$ time and we need at most m steps.

The rest of the paper is organized as follows. We introduce the notation and state the optimization problem in Section 2. In Section 3 we review the connection between agony and eulerian subgraphs. In Section 4 we introduce our optimization algorithm. We discuss the related work in Section 5 and present experimental evaluation in Section 6. Finally, we conclude the paper with remarks in Section 7.

2 Preliminaries and problem statement

Throughout the whole paper we assume that we are given a directed graph $G = (V, E)$. We will denote the number of vertices by $n = |V|$ and the number of edges by $m = |E|$. *All* graphs in this paper are directed and have the same vertices V. Given a graph $H = (V, F)$, we will write $E(H) = F$.

In this paper our goal is to discover a hierarchy among vertices in a graph G. That is, assume that we are given a graph $G = (V, E)$ and our goal is to discover a partition of vertices $\mathcal{P} = P_1, \ldots, P_k$, such that $P_i \cap P_j = \emptyset$ and $\bigcup_{i=1}^{k} P_i = V$, optimizing a certain quality score which we will define later. It will be more convenient to express this partition using a rank function, that is, our goal is to construct a function $r : V \to \mathbb{Z}$ mapping each vertex to an integer. We can easily construct a partition from this rank function by grouping the nodes mapping to the same value together.

Our next step is to define the quality score.

Given a rank function r, we say that an edge $e = (u, v) \in E$ is *forward* if $r(u) < r(v)$. Similarly, we say that $e = (u, v) \in E$ is *backward* if $r(u) \geq r(v)$. Note that the inequality is strict for forward edges.

As our goal is to discover hierarchy in G, in an ideal partition all edges are forward. This is only possible if G is a DAG which is rarely the case in practice. Consequently, we need a quality score that would penalize the backward edges. Given a rank r we define the *agony* of an edge $(u, v) \in E$ to be

$$q((u, v), r) = \max(r(u) - r(v) + 1, 0) \quad .$$

The agony for forward edges is 0 while the agony for backward edges is the difference between ranks plus 1. Note that the edges within the same block are penalized by 1.

Given a graph G and a rank r we define the agony of the whole graph to be the sum of individual edges,

$$q(G, r) = \sum_{e \in E} q(e, r) \quad .$$

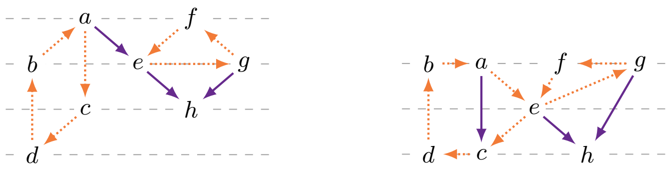


Fig. 1: Toy graphs. Dotted edges represent the eulerian subgraph. Ranks are represented by dashed grey horizontal lines.

Example 1. The agony of the left graph given in Figure 1 is equal to

$$q((b, a)) + q((d, b)) + q((e, g)) + q((g, f)) = 2 + 3 + 1 + 2 = 8 \quad .$$

The agony of the right graph is equal to

$$q((b, a)) + q((d, b)) + q((c, d)) + q((e, g)) + q((g, f)) = 1 + 3 + 1 + 2 + 1 = 8 \quad .$$

We can now state the main optimization problem of this paper.

Problem 1. Given a graph G find a rank function r minimizing agony $q(G, r)$.

Graph $H = (V, F)$ is called *eulerian* if the out-degree of each vertex is equal to its in-degree,

$$|\{u \in V; (v, u) \in F\}| = |\{w \in V; (v, w) \in F\}| \quad .$$

In the literature, H is sometimes required to be connected but here we do not impose this constraint.

Example 2. An example of eulerian subgraph in the left graph of Figure 1 consists of (b, a), (a, c), (c, d), (d, b), (f, e), (e, g), and (g, f).

An example of eulerian subgraph in the right graph of Figure 1 consists of (b, a), (a, e), (e, c), (c, d), (d, b), (f, e), (e, g), and (g, f).

Given a graph G we say that H is a *maximum* eulerian subgraph G if H is an eulerian subgraph of G and has the highest number of edges among all eulerian subgraphs of G. This graph is not necessarily unique. For notational simplicity, we require that G and H have the same vertices, $V(H) = V(G) = V$. This restriction does not impose any difficulties since we can always add missing vertices as singletons to H.

Given a graph G we say that H is a *maximal* eulerian subgraph G if H is an eulerian subgraph of G and we cannot increase H by adding new edges without making it non-eulerian. Note that maximum eulerian subgraph is necessarily maximal but not the other way around. It is easy to see that H is maximal if and only if the remaining edges in G form a DAG.

As we see in the next section, the following optimization problem, that is, finding the maximum eulerian subgraph is closely related to optimizing agony.

Problem 2. Given a graph G find an eulerian subgraph H maximizing $|E(H)|$, the number of edges.

3 Agony and eulerian subgraphs

In this section we review the connection between agony and discovering maximum eulerian subgraph. In fact, they are dual problems. This connection allows us to develop our algorithm in the next sections.

To see the connection let us first write the agony optimization problem as an integer linear program, that is, our goal is to solve the following program.

$$\begin{aligned} \min \sum_{(u,v) \in E} p(u, v) & \qquad \text{such that} \qquad (1) \\ p(u, v) \geq r(v) - r(u) + 1 & \qquad \text{for all } (u, v) \in E, \\ p(u, v) \geq 0 & \qquad \text{for all } (u, v) \in E, \\ p(u, v),\ r(w) \in \mathbb{Z} & \qquad \text{for all } (u, v) \in E,\ w \in V \quad . \end{aligned}$$

The solution for Eq. 1 will contain the optimal rank function r and agony for individual edges $p(u, v)$.

Let us relax the program by dropping the integrality conditions, thus transforming the program into a standard linear program. The dual of this program is equal to

$$\begin{aligned}
&\max \sum_{(u,v)\in E} c(u,v) && \text{such that} && (2)\\
&c(u,v) \leq 1 && \text{for all } (u,v) \in E, &&\\
&\sum_{(u,v)\in E} c(u,v) = \sum_{(v,w)\in E} c(v,w) && \text{for all } v \in V, &&\\
&c(u,v) \geq 0 && \text{for all } (u,v) \in E \quad . &&\\
& && && (3)
\end{aligned}$$

Assume that we are given a feasible solution to a dual problem such that $c(u,v)$ are integral. The conditions imply that $c(u,v)$ is either 0 or 1. If we form a subgraph H by taking the edges for which $c(u,v) = 1$, then the equality condition implies immediately that H is eulerian. Consequently, the solution for the dual problem is at least as large as the number of edges in the maximum eulerian graph.

Since the primal solution is always larger than the dual solution we have the following proposition.

Proposition 1. *Assume that we are given a graph G. Let r be a rank function and let H be an eulerian subgraph. Then $|E(H)| \leq q(G,r)$. Moreover, if $|E(H)| = q(G,r)$, then r minimizes agony and H has the maximum number of edges.*

Proof. Let P be the solution of Eq. 1 and let D be the solution of Eq. 2. Primal-dual theory (see, for example, [13]) states that $D = P$. We now have $q(G,r) \geq P = D \geq |E(H)|$. If $|E(H)| = q(G,r)$, then this immediately implies $|E(H)| = q(G,r) = P = D$, proving the optimality of r and H. $\square$

The previous result only proves that *if* there is a rank function r whose agony corresponds to the number of edges in the eulerian subgraph H, then r and H are optimal. It does not guarantee that such solution exists. Gupte et al. [7] showed that such solution always exists. However, we do not need this result. Instead, in the next section we introduce an algorithm that finds r and H satisfying the conditions of Proposition 1 which immediately implies the optimality of r.

4 Algorithm for discovering agony

In this section we present our algorithm based on the results of previous section. As our first step, we characterize the difference between the agony of the current rank function and the number of edges in the eulerian subgraph. We then present an algorithm that minimizes this difference and by doing so leads to the optimal solution. Finally, we present a fast algorithm for discovering a maximal eulerian subgraph, an initialization step that is needed for our main algorithm.

4.1 Gap between agony and eulerian subgraphs

In order to characterize the gap between the scores we need several concepts.

Assume that we are given a graph G and let H be a maximal eulerian subgraph G. We say that a rank function r *conforms* H if all backward edges with respect to r are in H. Note that this is possible only if H is maximal, otherwise there will be at least one backward edge in $E(G) \setminus E(H)$.

We will express the gap as a sum of slacks. More formally, given a rank r we define the *slack* of an edge as

$$\mathit{slack}((u,v),r) = \max(r(v) - r(u) - 1, 0) \quad .$$

Slack of (u,v) will be positive only if the edge is forward and the rank $r(v)$ is at least $r(u) + 2$.

We saw in the previous section that the agony is always larger than the number of edges in an eulerian graph. We can express this difference under certain conditions using slacks.

Proposition 2. *Assume that we are given a graph $G = (V,E)$ and let $H = (V,F)$ be a* maximal *eulerian subgraph. Let r be a rank function of V conforming H. Then*

$$q(G,r) = |F| + \sum_{e \in F} \mathit{slack}(e,r) \quad .$$

Moreover, if the sum of slacks is 0, then r has the lowest possible agony.

Proof. Since H is an eulerian graph, we can partition H into s edge-disjoint cycles $C_1, \ldots, C_s$. Since backward edges are only in H we can write agony as

$$q(G,r) = \sum_{e \in F} q(e,r) = \sum_{i=1}^{s} \sum_{e \in C_i} q(e,r) \quad .$$

The agony of a single edge $e = (u,v)$ can be written as

$$\begin{aligned} q(e,r) &= \max(r(v) - r(u) + 1, 0) = r(v) - r(u) + 1 - \min(r(v) - r(u) + 1, 0) \\ &= r(v) - r(u) + 1 + \max(r(u) - r(v) - 1, 0) \\ &= r(v) - r(u) + 1 + \mathit{slack}(e,r) \quad . \end{aligned}$$

Summing the edges in a single cycle gives us

$$\sum_{e \in C_i} q(e,r) = \sum_{e=(u,v) \in C_i} r(v) - r(u) + 1 + \mathit{slack}(e,r) = |C_i| + \sum_{e \in C_i} \mathit{slack}(e,r) \quad .$$

Since the cycles are edge-disjoint, we get the first result of the proposition. If the sum of slacks is 0, then the the agony $q(G,r)$ is equal to the number of edges in eulerian subgraph. Proposition 1 now implies that r is optimal and H is in fact a maximum eulerian subgraph. □

Example 3. Consider the left graph in Figure 1. The current agony is equal to 8 and the size of the current eulerian subgraph is equal to 7. There is one slack edge, namely $\mathit{slack}((a,c),r) = 1$. On the other hand, the right graph in Figure 1 has agony of 8 which is equivalent to the number of edges in the eulerian subgraph. There are no slack edges.

4.2 Algorithm for computing agony

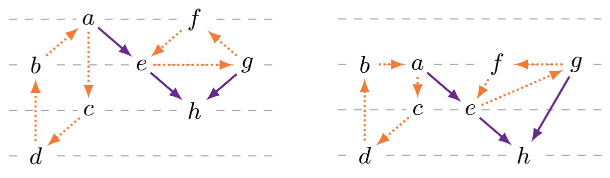
We are ready to describe the algorithm. Assume that we are given a graph G and assume that we have obtained a maximal eulerian subgraph and a rank r that conforms H. We will describe later how to obtain the initial H and r.

Proposition 2 states that r is optimal if there are no edges with slack in H. Assume there is one, say (p, s). We begin the algorithm by increasing the rank of p so that (p, s) has no slack. This may result that some of the edges outside H become backward, hence we will increase the rank of the end point of each new backward edge to make sure that there are no new backward edges. In addition, some of edges H may obtain more slack, hence we will also increase those vertices. These increases may require additional increases for other vertices and we keep doing this until either there are no more increases needed. If we do not encounter s during this algorithm, then we have successfully reduced agony by the $\mathit{slack}((p, s), r)$. Otherwise, we will show that we can modify H such that the number of edges in increased.

The visiting order of vertices is important in order to guarantee that the algorithm runs in $O(m)$ time. We will show that we can guarantee the running time if we keep the vertices in a priority queue based on how much we need to increase their rank, larger increases first.

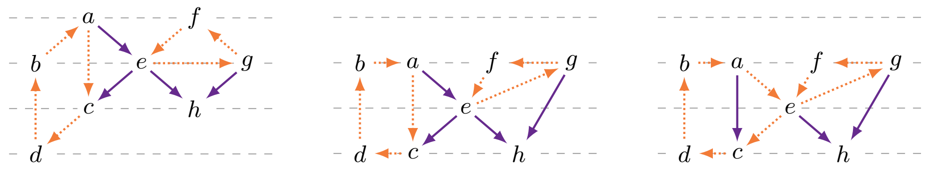
The pseudo-code for the algorithm is given in Algorithm 1. The algorithm takes as an input the underlying graph G, current maximal eulerian subgraph H and conforming r, and an edge $(p, s) \in E(H)$ with positive slack. The algorithm outputs a new subgraph H' and a new rank function r'.

Case 1: we can increase $r(a)$ without increasing $r(c)$



(a) input graph (b) final graph

Case 2: we cannot increase $r(a)$ without increasing $r(c)$



(c) input graph (d) after increasing ranks (e) final graph

Fig. 2: Two examples of applying Relief for (a, c). Dotted edges represent the eulerian subgraph. Ranks are represented by dashed grey horizontal lines.

Algorithm 1: Relief, given an maximal eulerian subgraph H and a conforming rank r, computes a new subgraph H' and a new rank function r' such that the agony or r is closer to the number of edges in the subgraph.

```
input  : underlying graph $G$, current maximal eulerian subgraph $H$, current
         rank function $r$, $(p, s) \in E(H)$ an edge with positive slack
output : updated maximal eulerian subgraph and new rank function
$F \leftarrow E(H)$;
$r' \leftarrow r$;
$t(v) \leftarrow 0$ for all $v \in V$ {how much we need to increase $v$}
$t(p) \leftarrow r(s) - r(p) - 1$;
add $p$ to $S$ with priority $t(p)$;
while $S$ is not empty do
    $u \leftarrow$ pop first element from $S$;
    $r'(u) \leftarrow r'(u) + t(u)$;
    foreach $(u, v) \in E \setminus F$ do
        if $r'(v) \leq r'(u)$ then
            $t \leftarrow r'(u) + 1 - r'(v)$;
            if $t > t(v)$ then
                $t(v) \leftarrow t$;
                add $v$ to $S$ with priority $t$, update $v$ if $v \in S$ already;
                $parent(v) \leftarrow u$;
    foreach $e = (w, u) \in F$ do
        if $slack(e, r') > slack(e, r)$ then
            $t \leftarrow slack(e, r') - slack(e, r)$;
            if $t > t(w)$ then
                $t(w) \leftarrow t$;
                add $w$ to $S$ with priority $t$, update $w$, if $w \in S$ already;
                $parent(w) \leftarrow u$;
    if $slack((p, s), r') > 0$ then
        $O \leftarrow$ edges in $E$ along the path from $s$ to $p$ using $parent$;
        $F \leftarrow (F \setminus O) \cup (O \setminus F)$;
        delete $(p, s)$ from $F$;
    return $(V, F)$, $r'$;
```

Example 4. Consider the graph given in Figure 2a. The eulerian subgraph is marked with orange dotted edges and the current rank function is represented by the dashed grey lines. Edge (a, c) has a slack of 1. Consider applying Relief on edge (a, c). The algorithm first increases $r(a)$. Edge (a, e) is no longer a forward edge, hence we need to increase e. This in turns transforms edge (e, h) into backward and increases the slack of (f, e). Ranks for both vertices are also increased. No other modifications are needed and the final graph is given in Figure 2b.

Now consider the graph given in Figure 2c and apply Relief on edge (a, c). As in previous case, e, f, and h are increased, but in addition c. Note that we did not manage to reduce the slack between a and c. However, if travel back along the *parent* links, $\mathit{parent}(c) = e$ and $\mathit{parent}(e) = a$ we obtain a path from c to a. By replacing (a, c) with these edges in the eulerian subgraph we obtain a new subgraph that has more edges.

The previous example showed the two possible outcomes for Relief, in both cases we reduce the slack. The following proposition states that this holds in general, that is, the new H and r are valid and that the difference between the costs is smaller.

Proposition 3. *Assume that we are given a graph G. Let $H = (V, F)$ be a maximal eulerian subgraph of G and let r be a rank function conforming H. Assume that there is an edge $(p, s) \in E(H)$ such that $\mathit{slack}((p, s), r) > 0$. Let $H', r' = \textsc{Relief}(H, r, G, p, s)$. Then H' is a maximal eulerian subgraph of G, r' is conforming H', $\max_{e \in E(H')} \mathit{slack}(e, r') \leq \max_{e \in E(H)} \mathit{slack}(e, r)$, and*

$$|\{e \in E(H') \mid \mathit{slack}(e, r') > 0\}| < |\{e \in E(H) \mid \mathit{slack}(e, r) > 0\}| \quad .$$

In order to prove this result we need the following lemma.

Lemma 1. *Each vertex visited at most once during* Relief. *Order the visited vertices based on their visiting order, say u_k. Let $t_k = t(u_k)$, where $t(u_k)$ is the priority of u_k at the time when u_k is visited. Then $t_{k+1} \leq t_k$.*

Proof. We will prove by induction over the iteration of Relief that once a vertex u has been removed from S it will never be added again to S and the priorities of newly added vertices into S during processing u is at most $t(u)$.

Assume that this holds for $k - 1$ first iterations, and let $u = u_k$ be a vertex that is visited during the kth iteration. Since S selects elements with the highest priorities, the induction assumption implies that $t_{i+1} \leq t_i$ for $i = 1, \ldots, k - 2$.

Let $(u, v) \in E \setminus F$. Let $t = r'(u) + 1 - r'(v)$. Since u is visited for the first time, we must have $r'(u) = r(u) + t(u)$ which implies that $t = t(u) + r(u) + 1 - r'(v) \leq t(u)$, where the inequality holds since (u, v) is a forward edge w.r.t. r. If v is added in S, then its priority is at most $t(u)$. This proves the second part of the induction step. On the other hand, if v has been already visited, say $v = u_j$, then $t_j = t(v) \geq t_k$ and v will not be added into S.

A similar argument can be made for the edges in F.

This proves the induction step and the lemma as the first step is trivial. □

Proof (of Proposition 3). Let us consider two separate cases. In Case 1, s remains unvisited while in Case 2 we visit s.

Case 1: Assume that we do not visit s. In such case, $H' = H$, hence H' is maximal.

We need to first show that edges in $E \setminus F$ remain forward. Whenever we increase the rank of u we check that none of the edges in $E \setminus F$ are backward. Assume there is one, say (u, v). If v is already visited, then Lemma 1 states that $t(v) \geq t(u)$. Since each vertex is visited only once, this implies that $r'(v) = r(v)+t(v)$ and $r'(u) = r(u)+t(u)$. This is a contradiction since (u, v) is a forward edge w.r.t. r. Hence, either v is not visited or is in S. Either way, we will increase $r'(v)$ so that v will become a forward edge at some point.

Using similar argument, we see that the slack of edges in F is not increased. Since the edge (p, s) is no longer a slack edge and we do not increase slack of any other edges, we have proved the proposition for Case 1.

Case 2: Assume that we have visited s. Write $F' = E(H')$.

Let us first argue that we can reach p by using the *parent* links. Lemma 1 implies that each vertex is visited only once which guarantees that *parent* links form a tree whose root is p.

Using the same argument as in Case 1, we see that forward edges in $E \setminus F$ remain forward edges and the slackness of edges in F is not increased. Moreover, one can easily show that (p, s) and the edges in $O \cap F$ are also forward edges. This means that $E \setminus F'$ contains only forward edges. This means that r' conforms H' and $E \setminus F'$ form a DAG which is only possible when H' is maximal. It is easy to see that H' is also eulerian.

Let $O_1 = O \cap (E \setminus F)$. For any edge $(u, v) \in O_1$, we must have $r'(v) = r'(u)+1$, otherwise $parent(v) \neq u$. This shows that the slack of the new edges is 0. Since (p, s) is removed from H' and we do not increase slack of any other edges, we have proved the proposition for Case 2. □

Our next step that this single iteration is linear in the number of edges.

Proposition 4. *The running time of* Relief *is* $O(m + slack((p, s), r))$.

Proof. Since each vertex is visited only once (Lemma 1) we will consider each edge only once. Hence, the inner for-loops are executed m times at most. Since the priorities of vertices are integers, we can implement the priority queue by storing each vertex into an array of $slack((p, s), r) - 1$ linked lists. Inserting or updating a vertex will take a constant time. Since the new priorities will always be smaller or equal, obtaining the maximum element takes $O(slack((p, s), r))$ of *total* time due to the fact that we need to possibly check some empty linked lists. This proves the proposition. □

Alternatively, we can implement the priority queue as a heap which gives the running time to be $O(m \log n)$.

In practice, we also apply the following speed-up. We monitor $t(s)$ constantly and we visit only those vertices that have larger priority. Since, Lemma 1 states that the priorities are non-increasing, we simply stop the main loop once we encounter a vertex with the same priority as $t(s)$. In addition, once we are done we backtrack rank of each visited vertex by $t(s)$. If s is not visited, then $t(s)$ remains 0 and this speed-up has no effect. However, if s is inserted in the stack S, we will prune vertices that have the same or lower priority than S. We ignore any vertex that should be lowered by at most $t(s)$. This may transform some forward edges into backward edges but we counter this by lowering the rank of the already visited vertices by $t(s)$. This implies that the forward edges remain forward and the arguments done in proof of Proposition 3 are valid.

We are now ready to state the main loop, given in Algorithm 2, which applies Relief to the edge with the largest slack.

Algorithm 2: MinAgony, given a graph G, a maximal eulerian subgraph H and a rank function r conforming H, finds a rank function optimizing agony

```
  input  : underlying graph $G$, maximal eulerian subgraph $H$, rank function $r$
  output : optimal maximal eulerian subgraph and rank function
1 while $q(G, r) > |E(H)|$ do
2     $(p, s) \leftarrow$ an edge in $E(H)$ with largest slack;
3     $H, r \leftarrow$ Relief$(H, r, G, p, s)$;
4 return $H$, $r$;
```

Our next step is to show that we need to call Relief at most $O(m)$.

Proposition 5. *Assume a graph G, a maximal eulerian subgraph H and a rank function conforming H such that $slack(e, r) \leq m$ for any edge $e \in H$. Then* MinAgony(G, H, r) *takes $O(m^2)$ time.*

Proof. Proposition 3 states that each call reduces the number of slack edges by at least 1. There can be at most m slack edges. Hence, the number of Relief calls is at most m. Since $slack(e, r) \leq m$ at the beginning and Proposition 3 states that slack is never increased, Proposition 4 implies that calling Relief takes $O(m)$ time. This completes the proof. □

Assume that we are given H, a maximal eulerian subgraph of G. Then $E(G) \setminus E(H)$ is a DAG, and any topological order will provide a rank function that is conforming with H. In this paper, we use a rank function, where we first remove all source vertices simultaneously from the DAG and assign them the same rank. We continue this until DAG is empty. The largest rank in this case is at most n, this also bounds the slack and consequently the conditions in Proposition 5 are satisfied.

4.3 Discovering maximal eulerian subgraph

Our final step is to discover a maximal eulerian subgraph. This can be done naively by running a DFS, finding and a removing a cycle and repeating until no cycles are left. This gives us running time of $O(m^2)$. A more sophisticated approach can be done with a single DFS, given in Algorithm 3.

CycleDFS starts with DFS and the moment it discovers a back edge, it finds a corresponding cycle. The algorithm proceeds by deleting the cycle and backtracking to the first vertex of visited cycle. The following proposition shows that the algorithm indeed finds a maximal eulerian subgraph.

Algorithm 3: CycleDFS, discovers a maximal eulerian subgraph.

```
input  : $G$, directed graph
output : $F$, edges corresponding to a maximal eulerian subgraph
while $V \neq \emptyset$ do
    $S \leftarrow$ any vertex in $V$;
    while $S \neq \emptyset$ do
        $u \leftarrow$ first vertex in $S$;
        if there is $(u, v) \in E$ then
            if $v \in S$ then
                $O \leftarrow (u, v)$ and the path from $v$ to $u$ along $S$;
                $F \leftarrow F \cup O$;
                delete $O$ from $G$;
                pop vertices from $S$ until the last vertex is $v$;
            else
                push $v$ to $S$;
        else
            pop $u$ from $S$;
            remove $u$ from $G$;
return $F$;
```

Proposition 6. CycleDFS *discovers maximal eulerian subgraph.*

Proof. F consists of edge-disjoint cycles, and by definition is eulerian. Assume that F is not maximal, that is, there is a cycle C. Let u be the first vertex in C that is deleted from G. Let e be the outgoing edge from u in C. By definition, e is not added in F. This implies that when we delete u from G, e is still present in G which is a contradiction since we only delete vertices with no outgoing edges. □

As a final step we show that CycleDFS runs in linear time.

Proposition 7. CycleDFS *executes in $O(m)$ time.*

Proof. During a single iteration of the inner while-loop we either delete x edges or push a vertex into a stack. Hence, the total running time is bounded by the number of edges deleted plus the number of pushes. Since each edge can be deleted only once, the first term is bounded by m. The number of times we will push a vertex u into S is bounded by the in-degree of u plus 1. Consequently, the number of pushes we will do in total is $O(n + m)$, which proves the result. □

5 Related work

From algorithmic point of view, the relation between our approach and the algorithm given by Gupte et al. [7] is intriguing. Both methods are based on primal-dual techniques, that is, they rely on the relationship between the primal problem, minimizing agony, and the dual problem, maximizing the eulerian subgraph. Gupte's algorithm is essentially an instance of the primal-dual algorithm, where one tries to improve the dual problem, in this case discovering maximum eulerian subgraph, until no improvement is possible. This improvement correspond to finding the negative cycle in a certain weighted graph, that is, a cycle whose sum of weights is negative. Currently the best algorithm for discovering negative cycle needs $O(nm)$ time [1] and this can be achieved with a Bellman-Ford algorithm [3]. Since we need m iterations at most, the computational complexity of this approach is $O(nm^2)$.

On the other hand, our approach is also an instance of the primal-dual algorithm. Especially, both algorithms improve the current eulerian subgraph. The difference is that while Gupte's algorithm searches the improvement by transforming the problem into discovering negative cycles, we discover the improvement in several calls of RELIEF. During each call of RELIEF if we have not able to find a new improvement for the eulerian subgraph, then we are able to improve the primal problem, that is, minimizing agony. In other words, while searching for improvement for the eulerian subgraph, we are able to use intermediate calculations to minimize the agony. This allows us to achieve a better computational complexity of $O(m^2)$.

The agony of a single edge is chosen very carefully. For example, if we choose agony to be 1 for every backward edge, then the problem is related to FEEDBACK ARC SET, where the goal is to discover a directed acyclic graph H. from a given directed graph G such that $E(G) \setminus E(H)$ is minimized. This problem is not only **NP**-hard, it is also **APX**-hard with a coefficient of $c = 1.3606$ [4]. There is no known constant-ratio approximation algorithm for FAS and the best known approximation algorithm has ratio $O(\log n \log\log n)$ [6].

Next, we highlight some of the existing methods for discovering hierarchies. Maiya and Berger-Wolf [11] suggested a statistical model where the probability of an edge is high between a parent and a child. To find the hierarchy they employ a greedy heuristic. Clauset et al. [2] studied discovering hierarchy in undirected graphs, where given a dendrogram, the probability of an edge between two vertices is based on Erdős-Rényi model, with a probability depending on the lowest common ancestor in the dendrogram. The authors then sample dendrograms using MCMC techniques. Macchia et al. [10] used agony to discover summaries of propagations based on traces. Jameson et al. [9] applied a model, where the likelihood of the the vertex dominating other is based on the difference of their ranks, to animal dominance data. Similar ideas has been used for ranking chess players by Elo [5]. Finally, hierarchy partitions vertices into groups, the top-level vertices having very different role than the bottom-level vertices. Assigning different roles to vertices have received some attention. Henderson et al. [8] consider assigning roles to vertices based on features while McCallum et al. [12] assigned topic distributions to individual vertices. An interesting direction for future work would be to study how hierarchy can be used for role mining in graphs.

6 Experimental evaluation

While we were able to improve the computational complexity of computing agony from $O(nm^2)$ to $O(m^2)$, the bound is still impractical even for graphs of modest size. Our next goal is to demonstrate empirically that this bound is in fact pessimistic and that we can compute the agony for large graphs.

In order to do so, we applied our algorithm for several large directed graphs, downloaded from Stanford Large Network Dataset Collection (SNAP).[1] We removed any edges of form (u, u) as they have no effect on the rank. The characteristics of the datasets are given in the first 2 columns in Table 1. In addition, to our algorithm we applied a baseline algorithm of Gupte et al. [7]. The algorithm requires a subroutine for detecting a negative cycle. We used Bellman-Ford algorithm with an additional speed-up, where after each iteration over the edges we check whether a cycle has been discovered. We implement both algorithms in C++ and performed experiments using a Linux-desktop equipped with a Opteron 2220 SE processor. The running times and detailed statistics are given in Table 1.

Table 1: Basic characteristics of the datasets and statistics from experiments. The 3rd column indicates the number of iterations, the 4th column indicates the slack of the starting point, the 5th depicts the final score. The running times for MinAgony and the baseline are given in 6th and 7th columns, respectively.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	iterations	gap	agony	time	baseline
Amazon	403 394	3 387 388	89 046	911 095	1 973 965	4h27m	–
Gnutella	62 586	147 892	1 907	150 851	18 964	45s	20m
EmailEU	265 214	418 956	27 679	500 177	120 874	2m	3h45m
Epinions	75 879	508 837	18 652	922 817	264 995	20m	1h40m
Slashdot	82 168	870 161	37 858	1 891 586	748 582	1h5m	7h3m
WebGoogle	875 713	5 105 039	164 708	4 110 696	1 841 215	2h32m	–
WikiVote	7 115	103 689	865	76 149	17 676	7s	1m

Our first observation is that the theoretical bound is indeed pessimistic. We are able to compute the agony for large networks in reasonable time. We spend 2.5 hours for computing agony for *WebGoogle*, a graph with over 5 million edges, and 4 hours for *Amazon*, a graph with over 3 million edges. For smaller graphs, the running time is significantly faster, either seconds or minutes.

The reason for this scalability is two-fold. First of all, the number of iterations, given in 4th column, is significantly lower than the number of edges. Secondly, Relief typically affects less than m vertices.

Our method performs better than the baseline for all datasets. We interrupted the baseline calculation after 24 hours for *Amazon* and *WebGoogle*.

[1] The datasets and their detailed descriptions are available at http://snap.stanford.edu/data/index.html

Finally, let us consider behaviour of agony and the size of the eulerian graph as a function of iterations. In order to do so we plot the evolution of scores, normalized by the final agony, in Figure 3.

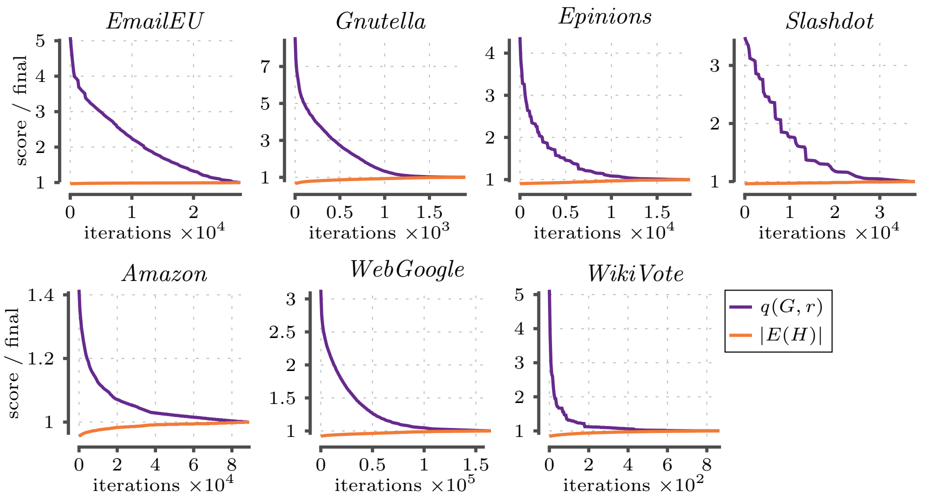


Fig. 3: Scores as a function of iteration. Each plot represents a single dataset. The upper line depicts the current agony score, normalized by the final score, as a function of a current iteration. The lower line depicts the current number of edges in the eulerian subgraph, normalized by the final score, as a function of a current iteration. Note that the x-axis is scaled.

We see that the initial agony is significantly larger than the final agony. For most datasets the agony drops quickly. For the largest datasets the algorithm achieves approximation ratio of 2 relatively quickly: for *WebGoogle* the algorithm achieves approximation ratio of 2 during the first 8% of iterations. This suggests that we can use the algorithm as any-time algorithm, stopping iterations early once we achieved acceptable approximation ratio. Note that since the optimal solution is at least as large as the current eulerian subgraph, we can at any time bound the approximation ratio of the current agony.

7 Concluding remarks

In this paper we introduced an algorithm for discovering hierarchy among vertices in a given directed graph. The hierarchy should minimize agony, the edges that violate the hierarchical structure. We show that our algorithm achieves computational complexity of $O(m^2)$ which is significantly better than the current bound of $O(nm^2)$. We also demonstrate that $O(m^2)$ is a pessimistic estimate of the running time and in practice the algorithm scales up for large networks.

There are several interesting directions for future work. An obvious and practical extension is to make edges weighted. Weighting edges will change the definition of the dual problem as we no longer are looking for maximum eulerian subgraph. On the other hand, integer weights can be viewed as multiple edges which should imply that the same framework can be applied. Another fruitful direction is to consider discovering hierarchies with constraints, such as the number of hierarchies or demanding that certain vertices have fixed ranks.

Acknowledgements. This work was supported by Academy of Finland grant 118653 (ALGODAN)

References

[1] Cherkassky, B.V., Goldberg, A.V.: Negative-cycle detection algorithms. In: ESA. pp. 349–363 (1996)

[2] Clauset, A., Moore, C., Newman, M.E.J.: Hierarchical structure and the prediction of missing links in networks. Nature 453(7191), 98–101 (2008)

[3] Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill Higher Education (2001)

[4] Dinur, I., Safra, S.: On the hardness of approximating vertex cover. Annals of Mathematics 162(1), 439–485 (2005)

[5] Elo, A.E.: The rating of chessplayers, past and present. Arco Pub. (1978)

[6] Even, G., (Seffi) Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multicuts in directed graphs. Algorithmica 20(2), 151–174 (1998)

[7] Gupte, M., Shankar, P., Li, J., Muthukrishnan, S., Iftode, L.: Finding hierarchy in directed online social networks. In: Proceedings of the 20th International Conference on World Wide Web. pp. 557–566 (2011)

[8] Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., Li, L.: Rolx: Structural role extraction &#38; mining in large graphs. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1231–1239 (2012)

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

[10] Macchia, L., Bonchi, F., Gullo, F., Chiarandini, L.: Mining summaries of propagations. In: IEEE 13th International Conference on Data Mining. pp. 498–507 (2013)

[11] Maiya, A.S., Berger-Wolf, T.Y.: Inferring the maximum likelihood hierarchy in social networks. In: Proceedings IEEE CSE'09, 12th IEEE International Conference on Computational Science and Engineering. pp. 245–250 (2009)

[12] McCallum, A., Wang, X., Corrada-Emmanuel, A.: Topic and role discovery in social networks with experiments on enron and academic email. J. Artif. Int. Res. 30(1), 249–272 (2007)

[13] Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc. (1982)