Retro-ingénierer les métriques topologiques dans les algorithmes de peer-ranking

Erwan Le Merrer, Gilles Trédan

Retro-ingénierer les métriques topologiques dans les algorithmes de peer-ranking

Erwan Le Merrer[1], Gilles Trédan[2]

[1]*Technicolor, France* [2]*LAAS/CNRS, France*

Détecter au plus tôt les utilisateurs importants dans les réseaux sociaux est un problème majeur. Les services de classement d'utilisateurs (peer ranking) sont maintenant des outils bien établis, par des sociétés comme PeerIndex, Klout ou Kred. Leur fonction est de "classer" les utilisateurs selon leur influence. Cette notion est néanmoins abstraite, et les méthodes algorithmiques de ces services pour obtenir ce classement sont opaques (les métriques et paramétrages utilisés sont des recettes internes dissimulées). Suivant le récent intérêt pour les outils permettant une plus grande transparence du web, nous proposons d'explorer le problème du retro-ingéniering de l'influence topologique dans ces services de classement. Comme ces services exploitent l'activité en ligne des utilisateurs pour inférer leur influence (dont leur connectivité sur les réseaux sociaux), nous proposons une approche permettant d'estimer précisément l'influence d'un ensemble de métriques topologiques (ou centralités) et leur taux de prise en compte dans le résultat final (classement utilisateur). Pour ce faire, nous modélisons l'algorithme de classement comme une boite noire, avec laquelle nous interagissons via des modifications topologiques, afin d'inférer quelles sont les centralités en jeu dans l'évolution du résultats de classement. Nous montrons que dans certains cas il est possible de déterminer quelles sont ces métriques, et ceci via des opérations sur la topologie par un utilisateur souhaitant connaitre ces paramètres.

Mots-clefs : Reverve-engineering ; centrality metrics ; social graphs ; ranking functions

As personal information concentrates in the cloud, so does the exploitation of this data. The need for an increased transparency in the functioning of web-services has recently arised, motivated by various use cases such as privacy or copyright control. For example, work such as [4] proposes to retrieve which piece of information of a user-profile triggered advertisement to that user. Goal is thus to infer the internals of black-box services provided by companies on the web. Klout or PeerIndex propose to rank users based on their behavior on social networks (using their social connectivity and activity). They nevertheless keep secret the algorithms and parameters used for this ranking. This motivated some users to try reversing their internals [2]. Sometimes information leaks about some ingredients of those hidden recipes ; CEO of PeerIndex admitted to use Pagerank (and thus graph topological-metrics), as a part of their ranking algorithm, to compute user influence.

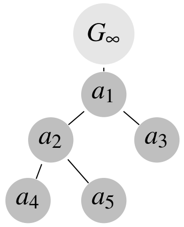
Reverse engineering such black-box is a challenging task. Indeed, in this web-service paradigm, the user only has access to the output of the algorithm, and cannot extract any side-information. Moreover, in many cases such as peer ranking services, the user is only aware and able to act on a limited part of the algorithm input. Motivated by this challenge for transparency, we thus ask the following questions : **can a user infer, from the results returned by such peer-ranking algorithms, what are the topological metrics in use, and if possible to what extent ? How difficult is such an attack ?**

We first introduce the service we consider and model our actions, before warming-up on a toy example. We then generalize our approach, and illustrate it on a concrete attack example. We finally give perspectives.

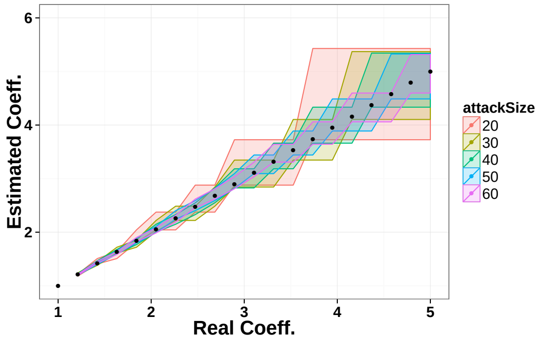
1 Model and Warm Up : Reversing the Use of a Single Centrality

We model the problem through an observable "ranking" black-box function f, that takes a graph as input (noted G_∞) and returns a ranking $>_r$ of all graph nodes such that : $\forall i,j \in V(G_\infty)^2, i >_r j$ iff "node i is more important than node j".

As warm up, let us assume that f is one of the following classic centralities $C_{base} = \{degree, eccentricity, betweeness, Pagerank, closeness\}$ [3]. To determine which is the used centrality, one attacker wants to build



(a) A small attack graph G_A, solving the single centrality reverse-engineering problem



(b) Reversing a f with hidden coefficients from 1 to 5, with various attack sizes (node creations)

a small attack graph G_A, attached G_∞ (then $G_\infty = G_\infty \cup G_A$, as being the giant social network), in order to reverse f. She has at her disposal means to edit the social graph, by adding nodes and edges. In practice, this consists in opening false accounts on the social network, and to link those (with e.g., "friend" requests). To get rid of the non-topological metrics that may be used in f, the attacker creates nodes $\in G_A$ that are strictly identical up to their connectivity (e.g., they have not Tweeted or posted any comment, etc).

Lemma 1. *A small attack graph, of* 5 *nodes, is sufficient to reverse a function* f *that is based on a single centrality in* C_{base}*, relatively to the other centralities in the same set* C_{base}*. Graph is depicted on Figure 1a.*

Proof Sketch. The proof requires showing that such G_A is able to discriminate the centralities considered in the set C_{base}. Consider graph $G_\infty \cup G_A$ on Figure 1a. G_A nodes are given the following ranking, for centralities in C_{base} : $< degree, [a_1 =_r a_2 >_r a_3 =_r a_4 =_r a_5] >$, $< eccentricity, [a_1 >_r a_2 =_r a_3 >_r a_4 =_r a_5] >$, $< betweenness, [a_1 >_r a_2 >_r a_3 =_r a_4 =_r a_5] >$, $< Pagerank, [a_2 >_r a_1 >_r a_4 =_r a_5 >_r a_3] >$, $< closeness, [a_1 >_r a_2 >_r a_3 >_r a_4 =_r a_5] >$. All rankings are indeed unique, thus allowing to discriminate the centrality used, by simple observation at G_A nodes' rankings produced by f. □

Note that this does not constitute a proof that a graph of size 5 is the minimal attack graph for such a problem and this centrality set C_{base}, or that this is the unique graph solving this problem. The interest of minimizing the size of the constructed attack graph is for the attacker twofold : first, constructing a bigger graph requires a longer time, especially if actions on the service platform are rate limited. Second, the bigger the attack, the easier it can be detected by the targeted service, which can then react accordingly.

For instance the graph $G_A \setminus n5$, of size 4 is not a solution, as *degree* and *betweenness* produce the same $[a_1 >_r a_2 >_r a_3 =_r a_4]$ ranking, as for both fringe nodes a_3 and a_4, *betweenness* is 0, and *degree* is 1. Sketch for a formal proof is to find of pairs of centralities over certain attack graphs that produce equivalent ranking to discard them, until reaching one particular graph where such a pair cannot be found. Following a possible set of centrality definitions, one attacker then designs an attack graph that is minimal.

2 A General Reverse-Engineering Approach

We now propose a reversing method for extending to a f that is a linear combination of centralities. Note that we also extend the notion of centrality to the one presented in [1], that is of any node-level measure.

Rationale The objective of the attack is to reverse-engineer f's internals, that is, to determine which centralities are used, and what is their relative importance in the linear combination that provides the ranking. G_∞ is not known to the attacker. As a social network user, she is able (through API calls) to create profiles, perform operations on these profiles, and observe the consequence of these operations on rankings in the peer-ranking service. As the space of possible centralities is theoretically infinite, we assume the attacker takes a bet on a list of d centralities in a set C, that are potentially involved in f.

In a nutshell, the attack proceeds as follows. The attacker leverages an arbitrary node a, already present in G_∞. She then creates d identical nodes and connects them to a ; those d nodes end up with the same ranking. She applies to each node a **different serie** of API calls (i.e., topological updates, attaching them one node for instance). After each serie of API calls, ranking of those nodes is expected to change. Based

```
   Data: $G_\infty$, a target node $a \in V(G_\infty)$, operations $\{u_1,\ldots,u_d\}$
   Result: An estimate of $\mathbf{h}$ (i.e., the vector containing the weight of each centrality in $f$)
1  //initialization
2  Let $\mathbf{k}$ be a vector of size $d-1$ initialized to 0;
3  for $1 \leq i \leq d$ do
4      //attach an attack node to an existing node, and conduce operations over it
5      Create node $a_i : V(G_\infty) \leftarrow V(G_\infty) \cup \{a_i\}$;
6      Add edge $(a_i,a) : E(G_\infty) \leftarrow E(G_\infty) \cup \{(a_i,a)\}$;
7      Apply $u_i(a_i)$;
8  W.l.o.g. assume $u_d$ is the operation with the highest impact (that is $\forall j < d, u_j <_r u_d$);
9  (Reorder otherwise);
10 for $i = 1$ to $d-1$ do
11     //identify operation thresholds
12     $\mathbf{k}_i \leftarrow max_{x \geq 1}(u_i^x(a_i) <_r u_d(a_d))$;
13 //J is the matrix where each element $(i,j)$ is the impact of $\mathbf{k}_i$ applications of $u_i$ on the $j^{th}$ centrality of node $a_i$,
   minus the impact of operation $u_d$ on $a_d$;
14 Let $J_{i,j} = c^j(u_i^{k_i}(a_i)) - c^j(u_d(a_d))$ ;
15 Set $J_{d,.} = 0^d$ ;
16 //find $\mathbf{h}$ s.t. $J.\mathbf{h} = 0$, thus is solution to the reverse-engineering of $f$
17 return Ker($J$)
```

FIGURE 2: General reverse-engineering algorithm : estimation of linear weight combination of f

on those observed changes, she is able to sort the impact of those calls, and thus to describe the impact of one given call by a composition of smaller effect calls. This allows her to retrieve the weights assigned by f to the d centralities in set C, by solving a linear equation system.

Lets consider the following image : imagine you have an old weighing scale (that only answers ”left is heavier than right” or vice-versa) and a set of fruits (say berries, oranges, apples and melons) you want to weigh. Since no ”absolute” weighing system is available, the solution is to weighs the fruits relatively to each other, for instance by expressing each fruit as a fraction of the heaviest fruit, the melon. One straightforward approach is to directly test how many of each fruit weigh one melon. This is the approach adopted here. However, to continue on the analogy, the problem here is that in general, we are not able to individually weigh each fruit (centrality). Instead, we have a set of d different fruit salads. This is not a problem since the composition of each salad (i.e., the impact of API calls) is known ; one has to solve a linear system using d different combinations of coefficients yielding an equal impact.

A reverse-engineering algorithm For demonstration purposes, let us assume that f relies on some centralities included in set C of size d. Let $\mathbf{c_i} \in \mathbb{R}^d$ be the d dimensional column vector representing each of the d involved centrality values for node i. We assume that f is **linear** in all directions (i.e., f is a weighted mean of all centralities) : $\exists \mathbf{h} \in \mathbb{R}^d$ s.t. $f(i) = \mathbf{c_i}.\mathbf{h}$. Reverse-engineering the topological impact over the final ranking thus boils down to find $\mathbf{h}$ (and therefore directly obtain f).

The attacker performs operations on G_∞ through API calls (or user operations). For instance, the following operation $u_1 : G_\infty(V,E), i \rightarrow^u (V \cup \{a\}, E \cup \{(i,a)\})$ simply adds a neighbor to i (we refer to it as $u_1(i)$). Each operation $u(i)$ impacts i's topological position, and therefore it's ranking by f. Let $\mathbf{u} \in \mathbb{R}^d$ be this impact on each centrality in C : $\mathbf{c}_{u(i)} = \mathbf{c}_i + \mathbf{u}$. We assume the attacker is able to find d different operations denoted $\{u_1,\ldots,u_d\}$ that respect the following properties : $i)$ she is able to determine the result of each u_i's impact on the target node's centrality values (i.e., compute $u_i^k(i), \forall i \leq d$, where $k > 0$ is the number of applications of the operation). And $ii)$ the d operations are mutually independent : each operation has a unique impact on set of considered centralities.

The attack proceeds as shown on Figure 2, where notations are defined. First, observe that by construction matrix J has not a full rank. The last update u_d is our reference against which we compare other updates. Line 12 records the maximum number of same u_i operation applications that lead to the same rank (or close) than a single u_d operation : that is the number of u_i operations needed to ”negate” the effects of u_d.

Consider a line i of J. Since at the end $u_i =_r u_d$ (or close), we have $(c_{a_i} + u_i^{\mathbf{k}_i}(a_i))\mathbf{h} = (c_{a_d} + u_d(a_d))\mathbf{h} \pm$

$u_i(a_i).\mathbf{h}$. Since by construction $c_{a_i} = c_{a_d}$, therefore we seek **h** s.t. $u_i^{\mathbf{k}_i}(a_i)\mathbf{h} - u_d(a_i)\mathbf{h} = 0$. Or as matrix notation : $J.\mathbf{h} = 0$: **h** is in the kernel of J.

Intuitively, the fact that we get infinitely many solutions ($\alpha.\mathbf{h}, \forall \alpha \in \mathbb{R}^+$) comes from our observation method : we are never able to observe actual scores, but rather rankings. Since multiplying **h** by a constant does not change the final ranking, any vector co-linear to, e.g., $\mathbf{h}/||\mathbf{h}||$ is a solution.

One important remark is that one cannot precisely claim that one centrality metric is not in use in C with our attack. Assume for instance that one centrality, say number of tweets, is 10^6 times less important than another, say degree. Then we will not be able to witness its effect unless we produce 10^6 tweets. And after $k < 10^6$ tweets, the only possible conclusion is : number of tweets is at least k times less important that degree. One can reasonably assume that such an imbalance in practice means that one service operator will not compute a possibly costly centrality to use it to such a low extent in f ; this thus makes our attack able to discard barely or not used centralities in C. To limit the detectability of the attack however, the maximal k_i after which u_i is considered to have a negligible impact should be carefully chosen.

Finally, one interesting complexity metric is the total number of calls to the API. Let $cost(u_i)$ be the number of calls issued by operation u_i. Then the total number of operations is $cost(u_d) + \sum_{i=1}^{d-1} k_i.cost(u_i)$. The precision of this estimation can be improved, but this approach is omitted due to space limitations.

An Illustration To illustrate our approach, let us assume a ranking function whose internals use a combination of c_1 : *clustering* centrality and c_2 : *degree*. W.l.o.g, assuming the coefficient for degree in **h** is $h_1 = 1$, we seek the corresponding coefficient $h_2 = h$. Let us consider the following two operations : u_1 that, applied to node a_1, adds a new node and connects it to all the neighbors of a_1 (i.e., modifies *clustering*), and u_2 that only adds a new node at each call (i.e., grows a star graph around a_2, thus modifying a_2's *degree*). The attacker can compute the value of $u_1^{k-1}(a_1)$ and $u_2^{k-1}(a_2)$ at any time.

We simulated the attack with a G_∞ being a $1,000$ nodes Barabási-Albert graph with an average degree of 5, estimating h using u_1 and u_2 operations. Figure 1b presents the obtained results : a point (x,y) means the real value of h is x and was estimated by the attack (using the algorithm presented on Figure 2) as y. Black dots plot the real coefficient values of h. Each colored area represents the estimated (reverse-engineered) coefficients, while each color represents the number of applications of the operations (refered to as the "attack size"). The larger the attack, the more precise the reverse-engineered results. We note that if the real values of coefficients to be estimated are bigger (e.g., 4 or 5 on the x-axis), estimations show lower precision (larger areas). Despite this remark, estimations are always correct.

3 Conclusion

The will for web-services transparency starts to trigger new research works. XRay [4] for instance proposes a correlation algorithm, that aims at inferring to which data input is associated a personalized output to the user. This Bayesian-based algorithm returns data that are the cause of received ads, while we seek in this paper to retrieve the internals of a hidden ranking function, in order to assess what is the effect of topology on the output peer-ranking. We have presented a general framework. Based on the centralities that might be used by the ranking function, there remain work for the attacker, to find operations (i.e., topological updates) that will make the reverse-engineering possible. We also find this to be an interesting challenge for future research.

Références

[1] S. P. Borgatti and M. G. Everett. A graph-theoretic perspective on centrality. Social networks, 2006.

[2] S. Golliher. How i reverse engineered klout score to an r2 = 0.94. blog post, 2011.

[3] D. Koschützki, K. A. Lehmann, L. Peeters, S. Richter, D. Tenfelde-Podehl, and O. Zlotowski. Network Analysis : Methodological Foundations, chapter Centrality Indices, pages 16–61. Springer, 2005.

[4] M. Lécuyer, G. Ducoffe, F. Lan, A. Papancea, T. Petsios, R. Spahn, A. Chaintreau, and R. Geambasu. Xray : Enhancing the web's transparency with differential correlation. In USENIX Security Symposium, 2014.