Perturb and Combine to Identify Influential Spreaders in Real-World Networks

Antoine J.-P. Tixier[1†], Maria-Evgenia G. Rossi[1], Fragkiskos D. Malliaros[2], Jesse Read[1], and Michalis Vazirgiannis[1]

[1]École Polytechnique, Palaiseau, France
[2]CentraleSupélec, University of Paris-Saclay and Inria Saclay, France
† corresponding author

Abstract. Recent research has shown that graph degeneracy algorithms, which decompose a network into a hierarchy of nested subgraphs of decreasing size and increasing density, are very effective at detecting the good spreaders in a network. However, it is also known that degeneracy-based decompositions of a graph are unstable to small perturbations of the network structure. In Machine Learning, the performance of unstable classification and regression methods, such as fully-grown decision trees, can be greatly improved by using Perturb and Combine (P&C) strategies such as bagging (bootstrap aggregating). Therefore, we propose a P&C procedure for networks that (1) creates many perturbed versions of a given graph, (2) applies a node scoring function separately to each graph (such as a degeneracy-based one), and (3) combines the results. We conduct real-world experiments on the tasks of identifying influential spreaders in large social networks, and influential words (keywords) in small word co-occurrence networks. We use the *k*-core, generalized *k*-core, and PageRank algorithms as our vertex scoring functions. In each case, using the aggregated scores brings significant improvements compared to using the scores computed on the original graphs. Finally, a bias-variance analysis suggests that our P&C procedure works mainly by reducing bias, and that therefore, it should be capable of improving the performance of all vertex scoring functions, not only unstable ones.

Keywords: Networks · Influential Spreaders · Ensemble Learning

1 Introduction

Graphs, or networks, are ubiquitous structures that can accurately model the interaction between the components of many natural and human-made complex systems [9]. Identifying nodes with good spreading characteristics is an important task with many real-world applications, from the study of human or animal epidemics [16], to viral marketing [24], social media analysis [40], expert finding [2], or keyword extraction from text corpora [38].

Graph degeneracy algorithms have recently been shown very effective at locating the good spreaders in a network [21, 27]. Moreover, these algorithms are affordable and can therefore give fast results in practice, even on very large graphs.

However, while graph degeneracy algorithms are very promising at influential spreader detection, they are unstable to perturbations such as the removal of a small fraction of edges from the network [1, 12]. Inspired by the bootstrap aggregating (bagging) Perturb and Combine (P&C) method which is well-known for improving the performance of unstable learners, we propose a P&C method for networks. Our procedure first creates many perturbed versions of a given graph, applies a node scoring function (such as a graph degeneracy algorithm) separately to each graph, and finally aggregates the results back into more robust scores.

Rigorous experiments conducted on the tasks of identifying influential spreaders in social networks and influential words (keywords) in word co-occurrence networks reveal that using the aggregated scores in lieu of the original scores allows to identify better spreaders. Furthermore, a bias-variance analysis reveals that P&C for networks is effective mainly by reducing bias (sometimes variance), and therefore might not only work well with unstable vertex scoring functions, but with all of them. The main contributions of our paper can be summarized as follows:

1. we propose what is, to the best of our knowledge, the first application of the Perturb and Combine (P&C) strategy to networks,
2. we conduct rigorous experiments on large social networks and small word co-occurrence networks, comparing three vertex scoring functions (k-core, generalized k-core, and PageRank). For all functions, results show that the aggregated scores allow to identify significantly better spreaders than the scores computed on the original graphs.
3. Our framework is general and can in theory improve the performance of any vertex scoring function. Also, our procedure is trivially parallelizable, so significant performance gains can be obtained at little extra cost compared to scoring ony the original network.
4. We define the bias and variance of a vertex scoring function, and explain through this lens why P&C for networks is effective.

The remainder of this paper is organized as follows: we provide some background about graph degeneracy, influential spreader detection, and P&C in Section 2. We then introduce our P&C framework for networks in Section 3. In Section 4, we present our experiments and report and interpret our results. Section 5 delivers the bias-variance analysis. Finally, we conclude in Section 6.

2 Related Work

2.1 Graph degeneracy

Let $G(V, E)$ be an undirected, weighted graph with $|V|$ nodes and $|E|$ edges. A core of order k (or k-core) of G is defined as the maximal subgraph of G in which every vertex v has at least degree k [36]. As shown in Figure 1, the k-core decomposition of G is the set of all its cores from 0 (G itself) to k_{max} (its main core). It forms a hierarchy of nested subgraphs whose cohesiveness and size respectively increase and decrease with k. The core number of a node is the highest order of a k-core subgraph that contains this node. That is, a node has core number k if it belongs to a k-core but not to a $(k + 1)$-core.

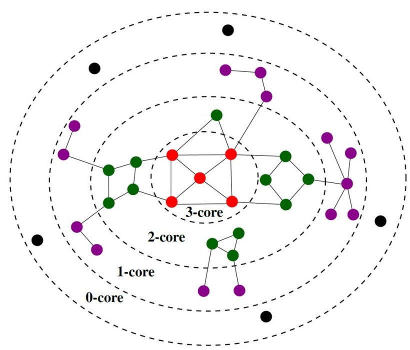

Fig. 1: Illustration of the k-core decomposition. Here, $k_{max} = 3$

The basic k-core algorithm is linear in the number of edges, but is *unweighted*, in that it uses as the degree of v the count of its neighbors. A generalized version of the k-core algorithm working in $O(|E| \log(|V|))$ for any local monotone vertex property function has been proposed [3]. By using the sum of the weights of the incident edges as the vertex property function (weighted degree), we obtain the *weighted* k-core decomposition of a graph. In the remainder of this paper, we use $cu(v)$ to denote the unweighted core number of node $v \in V$, and $cw(v)$ to refer to its weighted core number.

The unweighted and generalized k-core algorithms are the oldest and the most famous members of the broader *graph degeneracy* family of algorithms.

2.2 Influential spreader detection

Influential spreaders can be defined as those nodes in the network that can diffuse information to the largest part of the network in a given number of time steps. Identifying influential vertices in a graph is a very important task in many fields, ranging from national security to viral marketing, epidemiology, and even NLP (e.g., see subsection 4.2).

Influential spreader detection can be broken down into two subtasks: (1) identifying *individual* influential nodes, and (2) identifying a *group* of nodes that, by acting all together, are able to maximize the total spread of influence (the latter task is known as Influence Maximization). In this paper, we focus on the first task, i.e., the identification of *individual* influential nodes.

A straightforward approach towards finding effective spreaders in networks, is to consider *node centrality criteria* and in particular the one of *degree centrality*. In fact, several studies have examined how the existence of heavy-tailed degree distribution in real-world networks [11] is related to epidemic spreading [33] and robustness under cascades of failures in the Internet graph and power grids [30]. Nevertheless, it turns out that there are cases when node degree is not a good spreading predictor. Indeed, a node can have arbitrarily high degree, yet poorly connected neighbors. Consider for example a hub node located at the periphery of the network, as shown in Figure 2. In social networks, it was actually shown that influential spreaders are better identified via their core numbers rather than via their degrees, PageRank, or betweenness scores [21].

2.3 Perturb and Combine (P&C)

Instability. In Machine Learning, unstable algorithms are algorithms for which small changes in the training set result in large changes in predictions. These models are also known as *low bias-high variance algorithms* or *strong learners* [5]. Unpruned decision trees in classification and regression are good examples of such models. Indeed, adding or removing only a few observations to/from the training set may dramatically change the structure, and therefore, the predictions, of a fully grown decision tree.

P&C. Unstable learners can have their accuracy greatly improved by perturbing and combining. Of all the P&Cs strategies, **b**ootstrap **agg**regat**ing** (bagging) [4] is certainly the most popular, and one of the most effective. It is actually one of the key ingredients of the acclaimed Random Forest model [6], together with the random subspace method [15].
In the context of decision trees, bagging simply consists in training unpruned trees in parallel on bootstrap samples of the training set. Each bootstrap sample is a *perturbed version* of the training set which is generated by drawing from it with replacement until a set of the same size is obtained. To issue a forecast for a new observation, the predictions of all the trees are combined, e.g., through averaging in regression and majority voting in classification. Note that instability

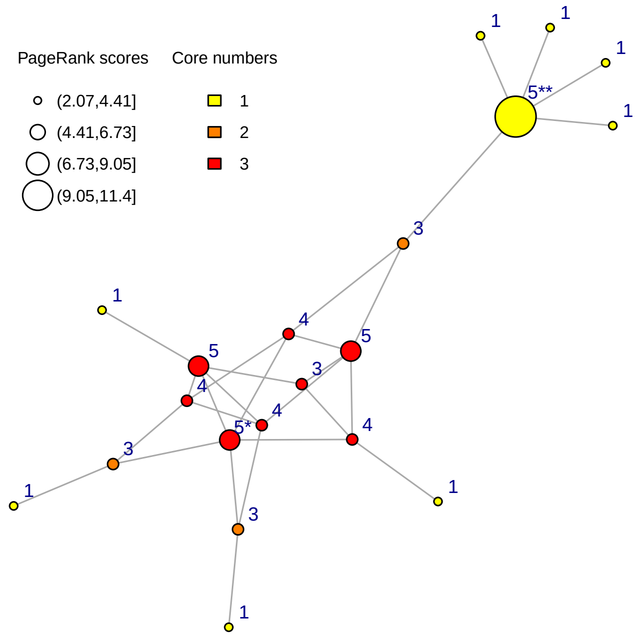

Fig. 2: Degree vs PageRank score vs core number. Node annotations indicate node degrees. Nodes $*$ and $**$ both have same degree (5) and high PageRank numbers (resp. in $(6.73, 9.05]$ and $(9.05, 11.4]$). However, node $*$ lies in a much more central location and is therefore a much better spreader, which is captured by its higher core number (3 vs 1) but not by degree or PageRank (the PageRank score of node $**$ is even greater than that of node $*$).

is mandatory for bagging to function well. For instance, bagging is not effective with nearest-neighbors approaches [4], which are very stable. Section 5 presents our P&C procedure for networks from a bias-variance perspective.

3 Perturb and Combine for Networks

3.1 Key idea

It was shown that the core decomposition of a network, which is used to locate the good spreaders in it (as discussed in subsection 2.2), is highly sensitive to small perturbations at the edge level such as edge addition and deletion [1]. This property is easy to verify in practice, as shown in Figure 4. Goltsev et al. also demonstrated that the removal of only a tiny fraction of vertices can vastly damage the k-core [12].

Motivated by the instability of the k-core decomposition, we posit that like unstable learners in Machine Learning, ***degeneracy-based node scoring func-***

tions, and more generally any unstable node scoring function, might benefit from using a perturb and combine strategy.

That is, one might be able to identify better spreaders by aggregating node scores computed on multiple perturbed versions of the original network rather than by using the scores computed on the original network. We therefore propose a P&C procedure for networks, illustrated in Figure 3 and summarized in Algorithm 1, that features three simple steps:

1. create M perturbed versions of a given graph,
2. separately apply a node scoring function to each graph,
3. combine the results.

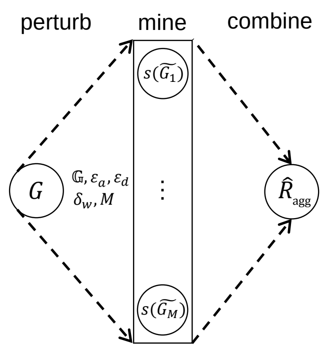

Fig. 3: Architecture of our P&C framework for networks. M perturbed versions of the original graph G are first created with a random graph model $\mathbb{G}$ and parameters $\varepsilon_a, \varepsilon_d$, and δ_w. A vertex scoring function s is then separately applied to each perturbed graph $\tilde{G}_m$. Scores are finally combined into an aggregated ranking $\hat{R}_{\text{agg}}$. Details about the perturb, mine, and combine parts can be respectively found in subsections 3.2, 3.3, and 3.4.

Algorithm 1 PC-NET (P&C FOR NETWORKS)

Input: original graph $G(V, E)$, $M \in \mathbb{N}$, vertex scoring function $s : V \mapsto \mathbb{R}$
Output: aggregated scores $s_{\text{agg}}(1), \ldots, s_{\text{agg}}(n)$ for the n nodes in V ($n = |V|$)

$B \leftarrow$ empty $M \times n$ array
for $m \in [1, \ldots, M]$ **do**
    $\tilde{G}_m \leftarrow$ PERTURB(G)
    $B[m, :] \leftarrow$ MINE($\tilde{G}_m$, s)
end for
$s_{\text{agg}}(1), \ldots, s_{\text{agg}}(n) \leftarrow$ COMBINE(B)
return $s_{\text{agg}}(1), \ldots, s_{\text{agg}}(n)$

Perturbation models have been widely used for generating graphs, describing them, and studying their behavior. However, this study is, to the best of our knowledge, the first time that graph perturbation is not only used for descriptive

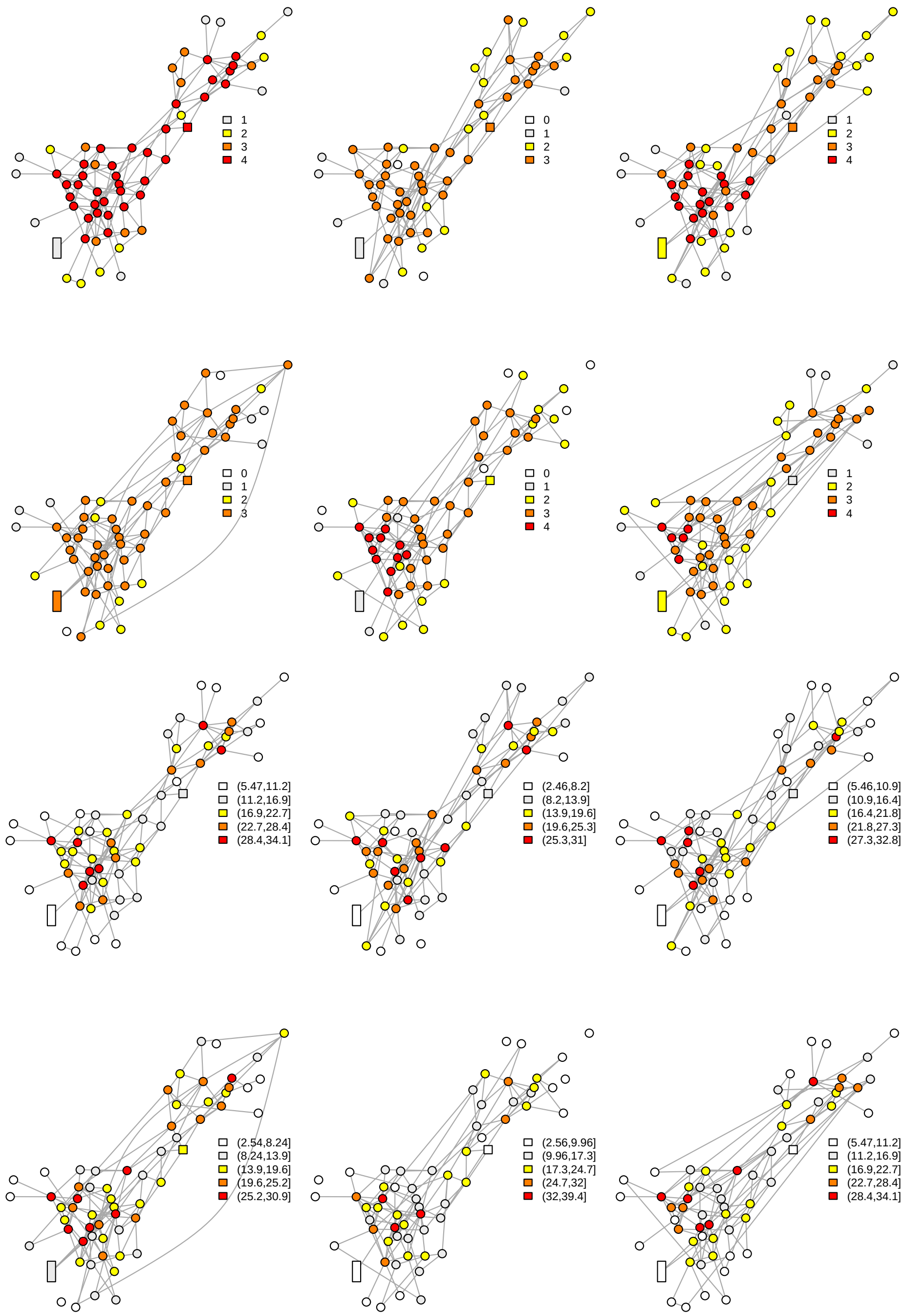

Fig. 4: **Top:** original graph (upper left corner) and 5 perturbed versions of it (generated by the edge perturbation strategy presented in subsection 3.2 with $\varepsilon_a = 0.1, \varepsilon_d = 0.3, \mathbb{G} = ER$, and $\delta_w = 0$). Node colors indicate unweighted k-core numbers. **Bottom:** same, but with PageRank. For k-core, we observe quite some variability in the core numbers across the different versions of the graph: the average cosine similarity between the 6 rankings is 93.29%. This illustrates well the unstable nature of the k-core vertex scoring function. PageRank rankings are only slightly more stable (average cosine similarity of 94.54%). The real-world network used in this example is the well-known *dolphins* network [26].

or generative purposes, but for the desirable side effects that it brings through subsequent *parallel mining* and *combination* (perturb and combine strategy).

Looking at Algorithm 1, it appears immediately that like Random Forest, PC-NET is a meta-algorithm that can trivially be parallelized (`for` loop in lines 2-5). The bagged scores therefore do not take more time to obtain than the original scores, provided that M workers are available. The only additional cost comes from the PERTURB step in line 3, but it can be implemented efficiently. Details about the PERTURB, MINE, and COMBINE parts (lines 3, 4, and 6) are provided in the next subsections (3.2, 3.3, and 3.4). Before going into the details of our approach, we illustrate and motivate it in what follows with a simple example.

Table 1: Original (s_G) and aggregated (s_{agg}) scores of the square and rectangle nodes in the network depicted in Figure 4, for the k-core function. $s_{\tilde{G}_m}$ refers to the m^{th} perturbed graph (the first 5 of which are shown in Figure 4). The aggregated score s_{agg} of a node is simply the average of its scores in each perturbed graph. *inf* denotes the influence as computed by the SIR model (see Figure 5).

	s_G	$s_{\tilde{G}_1}$	$s_{\tilde{G}_2}$	$s_{\tilde{G}_3}$	$s_{\tilde{G}_4}$	$s_{\tilde{G}_5}$	$s_{\tilde{G}_6}$	$s_{\tilde{G}_7}$	$s_{\tilde{G}_8}$	s_{agg}	*inf*
square	**4**	3	3	3	2	1	2	3	2	**2.38**	4.09
rectangle	**1**	1	2	3	1	2	1	1	2	**1.62**	2.76

Motivational example. Look at the square and rectangle nodes in Figure 4. In the original graph, the rectangle node is part of the 1-core, which is the lowest level in the hierarchy. However, we can see that it has direct access to the top level ($k = 4$) through its only connection. Should the rectangle node only have a couple of additional connections, it would probably benefit from its closeness to the main core and move upwards in the hierarchy. On the other hand, the square node is a member of the top core in the original graph, but a quick visual inspection reveals that it does not lie in the most central part of the network and is not strongly attached to the main core. Actually, with a degree of only 4, it is one of the weakest members of the top core ($k = 4$), i.e., removing one of its connections would suffice in downgrading that node.

By slightly perturbing the original network (adding and deleting a few edges at random) as shown in Figure 4, we are able to generate different configurations of the original network, which, as will be discussed in Section 5, can in essence be seen as drawing from some true (but unavailable) underlying graph of which the original network is a single snapshot. Applying the vertex scoring function separately to each graph and combining back the scores can therefore be seen as *estimating the true scores of the nodes based on more evidence*.

Going back to our example, while the rectangle node has a low score in the original network, it gets a higher score in most perturbed graphs. Through subsequent combination, P&C is thus able to capture that the true score of the rectangle node should probably be greater than its original score (1.62 > 1), as shown in Table 1. Conversely, P&C detects that the square node is not tightly connected to the main core of the original network (as in most perturbed graphs, it gets downgraded) and that therefore, its true score should probably be lower than its original score (2.38 < 4).

To sum up, while the square and rectangle nodes are very distant in the original ranking, with scores of 4 and 1 (respectively), the P&C procedure brings them closer in the aggregated ranking (2.38 and 1.62). This is much more faithful to the true ranking: indeed, the square and rectangle nodes have roughly the same influence, as shown in Figure 5.

3.2 Perturb

High-level framework. We used a flexible framework very similar to that used in [1]. It is a general model for edge-based perturbation of which most of the perturbation models found in the literature can be seen as special cases. Using only edge perturbation and not node perturbation is desirable in our case as combining the results (COMBINE step) is more straightforward if each node receives a score in each perturbed graph.
Let $G(V, E)$ be the original graph and $\mathbb{G}$ be a random graph model. The corresponding perturbation model $\Theta(G, \mathbb{G}, \varepsilon_a, \varepsilon_d)$ is defined as:

$$\mathbb{P}_\Theta[(u,v)] = \begin{cases} \varepsilon_a \mathbb{P}_\mathbb{G}[(u,v)], \text{ if } (u,v) \notin \mathrm{E} \\ \varepsilon_d \mathbb{P}_\mathbb{G}[(u,v)], \text{ if } (u,v) \in \mathrm{E} \end{cases} \tag{1}$$

where $\mathbb{P}_\Theta[(u,v)]$ is the probability of adding/deleting the edge (u,v), $\mathbb{P}_\mathbb{G}[(u,v)]$ is the probability of selecting the edge (u,v) according to the random graph model $\mathbb{G}$, and ε_a, ε_d are the probabilities of edge addition and deletion, respectively.

An interpretation can be given as follows: if the edge (u,v) already exists, then it is deleted with probability $\varepsilon_d \mathbb{P}_\mathbb{G}[(u,v)]$, and if not, it is added with probability $\varepsilon_a \mathbb{P}_\mathbb{G}[(u,v)]$. By XOR-ing the original graph G with one realization $\theta \sim \Theta(G, \mathbb{G}, \varepsilon_a, \varepsilon_d)$ of the perturbation model, we obtain the perturbed graph $\tilde{G} = G \oplus \theta$. Depending on the random graph model $\mathbb{G}$ used, we obtain a different perturbation scheme.

Edge weight awareness. To make our approach more flexible, and generalizable to weighted graphs, we use two variants of the above perturbation framework:

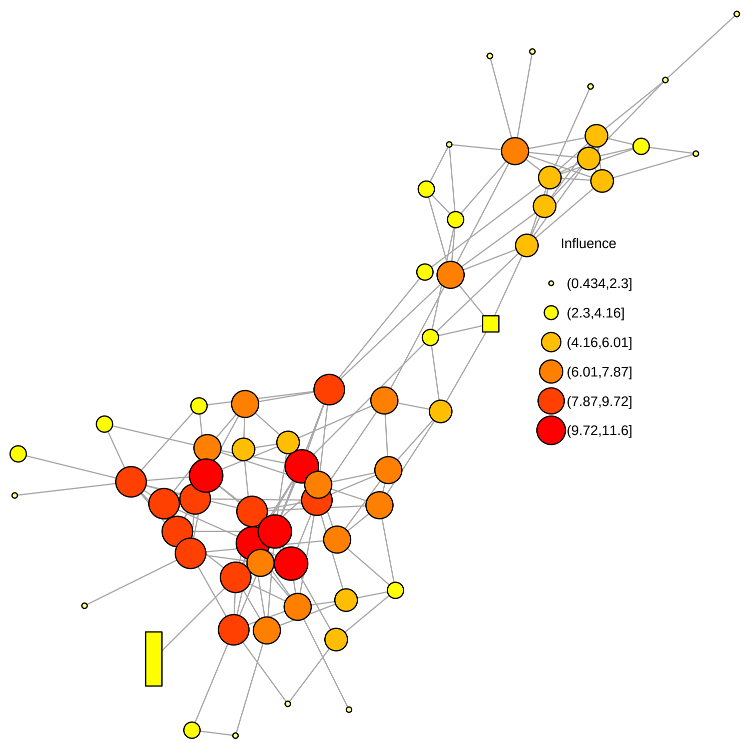

Fig. 5: Influence of the nodes of the *dolphins* network, as measured by the number of nodes infected at the end of a SIR epidemic triggered from each node, with 200 iterations, infection rate $\beta = 0.016$, recovery rate $\gamma = 0.8$, and epidemic threshold $\tau = 0.139$ (the SIR model is defined in subsection 4.1). Scores were averaged over 100 runs to account for the stochastic nature of the SIR epidemics. The exact influence of the square and rectangle nodes is equal to 4.09 and 2.76, respectively. Notice how degree centrality (number of connections) is not predictive of spreading efficiency. Rather, the best spreaders are located in the inner-cores of the network.

• the first scenario ($\delta_w = 0$) is exactly the one described above, i.e., it ignores edge weights during the perturbation phase. Edges can only be completely removed or created from scratch.

• In the second scenario ($\delta_w = 1$), the perturbation procedure accounts for edge weights. More precisely, a given edge (u, v) can be considered for addition even if it already exists, and can remain in the graph even if it was selected for deletion. In such cases, we simply increment (respectively, decrement) the weight of (u, v) by one standard deviation of all edge weights. Since edges can be selected multiple times, any edge whose weight becomes negative is deleted from the graph.

In the rest of this paper, the δ_w parameter will indicate which of the two variants above is being used. Note that in both scenarios, every time a new edge is added to the network, we sample its weight at random from the weights of the edges incident on u and v.

Random graph models. Plugging the Erdős-Rényi (**ER**) random graph model [10] into our framework returns the *uniform perturbation model*. A node is randomly drawn with replacement from V with probability $1/n$. On the other hand, using the Chung-Lu (**CL**) random graph model [8] gives the *degree assortative perturbation model*. A node is randomly drawn with replacement from V with probability proportional to its weighted degree. In that case, edges are more likely to be created/incremented and deleted/decremented between hubs. For both $\mathbb{G} = ER$ and $\mathbb{G} = CL$, self-edges are disregarded. That is, if we select two nodes u and v such that $u = v$, we discard the pair and select two other nodes.

3.3 Mine

Vertex scores can be returned by any node scoring function. However, since perturb and combine strategies in Machine Learning are most effective when used with *unstable* base learners, we experimented with unstable functions such as k-core and weighted k-core. For the sake of comparison, we also tried with the weighted PageRank algorithm [32], since PageRank rankings are known for being more stable to link-based perturbations [18, 31].

3.4 Combine

Following common practice in Machine Learning (e.g., bagging regression trees), we consider in this study the aggregated score of a node as the average of its scores in each of the M perturbed graphs.

4 Experiments

In this section, we present the experiments we conducted on two standard real-world datasets of social and word co-occurrence networks. In each case, we use three vertex scoring functions: unweighted k-core (cu), weighted k-core (cw), and weighted PageRank (pr). For each function, we compare the spreading performance of the nodes with highest scores in the original networks to that of the nodes with highest aggregated scores. In what follows, for each dataset, we introduce the networks it contains, detail our experimental setup, and finally report and interpret our results.

4.1 Social Networks

Networks. We used a set of three well-known, publicly available large scale social networks [25] which we briefly describe in what follows. Key descriptive statistics of these networks are also shown in Table 2.

- **Email-Enron** is an email communication network that was created based on the email exchanges among the members of the Enron Corporation. These data were made public by the Federal Energy Regulatory Commission during the Enron investigation [22]. It includes in total about 500K messages. Each node represents an email address and there is an edge between two nodes if the users they represent exchanged at least one email.
- **Epinions** is a who-trust-whom online social network of the general consumer review site `epinions.com`. Each member decides whether to trust the other members. From the trust relationships that are created together with the review ratings provided by the users, reviews are suggested to users.
- The **WikiVote** network was created by extracting all administrator elections and vote history data from Wikipedia. It contains data from $2,794$ elections with $103,663$ total votes and $7,066$ users ranging from the inception of Wikipedia to January 2008.

For the networks above, which are not weighted originally, we assigned as edge weights the maximum degree (number of connections) of the endpoints.

Experimental Setup. To measure the spreading influence of a node, we have to simulate a diffusion process. In this study, we used the Susceptible-Infected-Recovered (SIR) epidemic model [20], which is the most commonly used model in the domain of influential spreader detection. The SIR model is a discrete time model which assumes that at every step, each node in the network falls into one of the following mutually exclusive categories:

Table 2: Statistics of the real-world social networks used in this study. $cu_{\max}$, cw_{max} and pr_{max} denote the maximum k-core, weighted k-core, and PageRank numbers respectively, and $\tau = {}^{1}/_{\lambda_1}$ is the epidemic threshold of the graph [7] (where λ_1 is the largest eigenvalue of the adjacency matrix).

Network	$\|V\|$	$\|E\|$	*diameter*	cu_{max}	cw_{max}	pr_{max}	τ
Email-Enron	33,696	180,811	11	43	18,931	14.95	8.4×10^{-2}
Epinions	75,877	405,739	14	67	37,343	3.06	5.4×10^{-2}
Wiki-Vote	7,066	100,736	7	53	15,756	4.37	7.2×10^{-2}

- Susceptible (S): the individual is not yet infected, thus being able to get infected with the disease,
- Infected (I): the individual has been infected with the disease and is capable of contaminating its susceptible neighbors,
- Recovered (R): after an individual has been infected, it is considered as immune and is not able to get infected or to transmit the disease anymore.

As shown in Figure 6, every susceptible node can be contaminated with probability β (infection rate) and can recover with probability γ (recovery rate). Note that at any given time step, a node can only recover after he was given a chance to infect its neighbors. Initially, all the nodes in the network belong to S, except the one whose spreading performance is to be studied, which is the unique member of the I set. The process iterates until no new node gets infected for two consecutive steps.

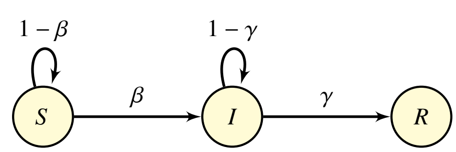

Fig. 6: Susceptible-Infected-Recovered (SIR) epidemic model state diagram. β and γ are the infection and recovery rates, respectively.

Following common practice [21, 27], we start N_e epidemics from each node in the trigger population to account for the stochastic nature of the SIR model. The results for each node are averaged over the N_e runs and again averaged over the trigger population to get a final performance score for a given vertex scoring function.

For the unweighted and weighted k-core functions, we set the trigger population to be the maximal k-core subgraph. Since we use averaging as our combination strategy, the aggregated scores may be real numbers. Therefore, we round them to the upper integer before selecting the top core. For PageRank, we set the trigger population to be the 100 nodes with highest PageRank scores.

We record, while the epidemic is unfolding, the number of nodes going from susceptible state to infected state at each time step. In the end, we also compute the total number of nodes that have been infected during the epidemic.

We set $N_e = 100$; the infection rate β close to the epidemic threshold of the network $\tau = \frac{1}{\lambda_1}$, where λ_1 is the largest eigenvalue of the adjacency matrix [7]; and the recovery rate γ to 0.8, as in Kitsak et al. [21].

We grid searched the following P&C parameters: $\varepsilon_a : \{0, 0.05, 0.1, 0.2\}$, $\varepsilon_d : \{0, 0.05, 0.1, 0.2\}$, $N : \{16, 64\}$, $\mathbb{G} : \{ER, CL\}$, and $\delta_w : \{0, 1\}$. Excluding the cases where $\varepsilon_a = \varepsilon_d = 0$, this made for 120 parameter combinations. The optimal parameter values were selected as the ones returning the greatest total number of nodes infected during the epidemic.

Results. Tables 3 to 5 show the number of newly infected nodes for the early time steps of the spreading process corresponding to the outbreak of the epidemic (steps 2, 4, 6, 8, and 10), and the total number of nodes that were infected during the epidemic (total), respectively for the unweighted k-core, weighted k-core and PageRank scoring functions.

We compare the average severity of the epidemic when started from the top nodes in terms of their scores on the original networks to that of the epidemic when started from the top nodes in terms of their aggregated scores for the best combination of P&C parameters shown in Tables 6, 7, and 8.

We can observe that for all three vertex scoring functions and for all three networks, using the aggregated scores to identify the nodes from which to trigger the epidemic systematically leads to a greater number of nodes infected during the epidemic. Moreover, the differences are substantial, comparable with the improvements reported in previous research [27].

We can also remark that even though the parameters were tuned to maximize the total number of nodes infected during the entire epidemic, nodes with high aggregated scores are better spreaders than nodes with high original scores even in the early stages of the diffusion process.

Finally, as shown in Figures 8 and 9, the rankings provided by P&C are of better quality than the rankings provided by the original scores, especially for the very top nodes. For the unweighted k-core function for instance, on the Enron corpus, the top 0.5% (169) nodes according to their aggregated scores contain 75.15% of the 0.5% best spreaders, compared to only 61.54% when using the

Table 3: Spreading performance comparison (number of nodes getting infected in the early stages of the epidemic and in total) for the unweighted *k*-core scoring function.

		Time Step					
Network	Scores	2	4	6	8	10	Total
EMAIL-ENRON	**aggregated**	**16**	**89**	**300**	**419**	**269**	**2,538**
	original	14	77	269	401	275	2,446
EPINIONS	**aggregated**	**8**	**34**	**110**	**245**	**317**	**2,436**
	original	7	30	100	224	301	2,330
WIKIVOTE	**aggregated**	**3**	**8**	**17**	**29**	**40**	**490**
	original	3	8	16	28	37	473

Table 4: Spreading performance comparison (number of nodes getting infected in the early stages of the epidemic and in total) for the weighted *k*-core scoring function.

		Time Step					
Network	Scores	2	4	6	8	10	Total
EMAIL-ENRON	**aggregated**	**26**	**141**	**407**	**445**	**226**	**2,724**
	original	20	110	345	433	253	2,628
EPINIONS	**aggregated**	**11**	**46**	**146**	**302**	**353**	**2,689**
	original	11	42	135	286	345	2,624
WIKIVOTE	**aggregated**	**5**	**12**	**24**	**39**	**50**	**612**
	original	4	9	18	31	42	513

Table 5: Spreading performance comparison (number of nodes getting infected in the early stages of the epidemic and in total) for the PageRank scoring function.

		Time Step					
Network	Scores	2	4	6	8	10	Total
EMAIL-ENRON	**aggregated**	**16**	**86**	**278**	**389**	**266**	**2,454**
	original	15	80	259	366	255	2,333
EPINIONS	**aggregated**	**11**	**42**	**132**	**276**	**336**	**2,598**
	original	11	41	127	267	326	2,545
WIKIVOTE	**aggregated**	**5**	**11**	**22**	**38**	**49**	**596**
	original	5	11	22	36	48	582

Table 6: Top 5 best P&C parameters for each scoring function (in terms of number of nodes infected during the epidemic), on the ENRON network. *cu*, *cw*, and *pr* respectively denote unweighted *k*-core, weighted *k*-core, and PageRank.

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**pr**	*cw*	*cu*
0	*ER*	64	0.2	0.2	**2454**	2641	2443
1	*ER*	16	0.2	0.2	**2454**	2613	2444
0	*ER*	16	0.05	0.2	**2447**	2663	2434
1	*ER*	64	0.2	0.2	**2446**	2684	2437
1	*ER*	16	0.05	0.2	**2445**	2649	2474

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**cw**	*pr*	*cu*
0	*CL*	64	0.2	0	**2724**	2225	2456
0	*CL*	64	0.1	0.1	**2718**	2343	2415
1	*CL*	64	0.2	0.2	**2717**	2360	2426
1	*CL*	64	0.1	0.1	**2710**	2364	2445
1	*ER*	64	0.05	0	**2709**	2333	2444

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**cu**	*pr*	*cw*
1	*ER*	64	0.2	0.05	**2538**	2349	2666
1	*ER*	64	0.2	0.1	**2529**	2419	2525
0	*ER*	64	0.1	0.1	**2521**	2409	2675
0	*ER*	64	0.05	0.05	**2513**	2422	2625
0	*ER*	64	0.1	0.05	**2508**	2321	2551

Table 7: Top 5 best P&C parameters for each scoring function (in terms of number of nodes infected during the epidemic), on the EPINIONS network. *cu*, *cw*, and *pr* respectively denote unweighted *k*-core, weighted *k*-core, and PageRank.

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**pr**	*cw*	*cu*
1	*ER*	16	0.2	0.2	**2598**	2638	2391
1	*CL*	64	0.2	0.05	**2597**	2654	2296
1	*ER*	64	0.2	0.2	**2596**	2652	2388
1	*ER*	16	0.1	0.2	**2596**	2631	2370
1	*CL*	64	0.2	0.2	**2593**	2607	2310

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**cw**	*pr*	*cu*
0	*ER*	64	0.1	0	**2689**	2553	2332
1	*CL*	64	0.05	0.05	**2673**	2543	2246
0	*ER*	64	0.2	0.05	**2673**	2552	2265
0	*ER*	64	0.1	0.2	**2672**	2565	2324
1	*CL*	64	0.1	0.1	**2672**	2573	2245

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**cu**	*pr*	*cw*
1	*ER*	16	0.2	0.05	**2436**	2574	2653
1	*ER*	16	0.2	0.1	**2436**	2581	2646
1	*CL*	64	0.2	0	**2423**	2573	2667
1	*ER*	16	0.2	0	**2419**	2557	2653
1	*ER*	16	0.2	0.2	**2391**	2598	2638

Table 8: Top 5 best P&C parameters for each scoring function (in terms of number of nodes infected during the epidemic), on the WIKIVOTE network. *cu*, *cw*, and *pr* respectively denote unweighted *k*-core, weighted *k*-core, and PageRank.

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**pr**	*cw*	*cu*
0	*ER*	16	0.2	0.2	**596**	523	474
0	*ER*	16	0	0.2	**595**	523	463
1	*ER*	64	0.05	0.2	**594**	537	465
0	*ER*	16	0.1	0.2	**593**	542	450
0	*ER*	16	0.2	0.05	**593**	529	461

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**cw**	*pr*	*cu*
0	*ER*	64	0.2	0	**612**	584	469
0	*CL*	64	0.2	0.05	**600**	574	440
0	*CL*	64	0.2	0.1	**589**	584	438
0	*CL*	64	0.1	0.1	**582**	573	453
0	*CL*	64	0.1	0.05	**578**	581	455

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**cu**	*pr*	*cw*
1	*CL*	16	0.05	0.05	**490**	575	532
1	*ER*	64	0.2	0	**488**	578	555
1	*CL*	16	0.1	0.05	**487**	581	530
1	*ER*	16	0.1	0	**487**	572	529
1	*ER*	16	0.05	0	**485**	576	530

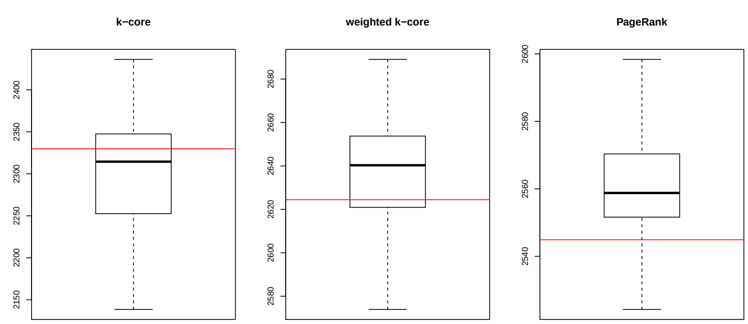

Fig. 7: For the Epinions network, distribution of the total number of nodes infected during the epidemic for the 120 combinations of P&C parameters. The severity of the epidemic by triggering it from the top nodes in terms of their scores in the original network is shown in red. The distributions look similar for the other networks.

original scores. The number of nodes corresponding to each percentage value are reported for each network in Table 9.

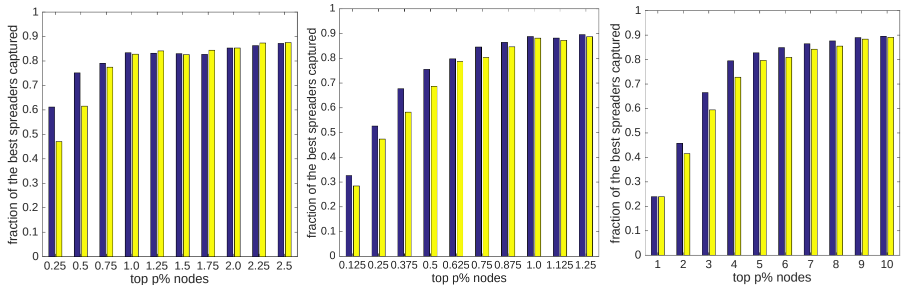

Fig. 8: Fraction of the p% best spreaders (as computed by SIR) captured by the top p% nodes according to the aggregated (first bars) and original (second bars) scores, on the Enron, Epinions, and WikiVote datasets (from left to right), for the **unweighted** ***k*-core** function.

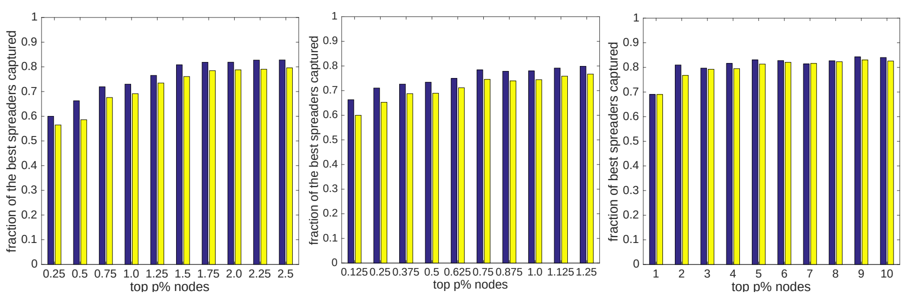

Fig. 9: Fraction of the p% best spreaders (as computed by SIR) captured by the top p% nodes according to the aggregated (first bars) and original (second bars) scores, on the Enron, Epinions, and WikiVote datasets (from left to right), for the **PageRank** function.

Discussion. Even though we observed some variability in PageRank rankings empirically (e.g., Figure 4), the fact that P&C improves the performance of PageRank was unexpected, as it has been suggested that PageRank is quite stable to edge-based perturbations [18, 31]. The implication could be that our P&C procedure can improve the performance of any node scoring function, not only unstable ones. Another explanation could be that unlike what previous research has shown [18, 31], PageRank features some level of instability. Why our method works well is further discussed in terms of bias and variance in Section 5.

Table 9: Number of nodes corresponding to each percentage value in Figures 8 and 9.

Enron	85	169	253	337	422	506	590	674	759	843
WikiVote	71	142	212	283	354	424	495	566	636	707
Epinions	95	190	285	380	475	570	664	759	854	949

Also, as can be seen in Figure 7 for the Epinions dataset, most combinations of parameters return aggregated scores that allow to identify better spreaders than the original scores, especially for weighted *k*-core and PageRank. Tables 6, 7, and 8 also show that the difference in performance is relatively small among the best combinations of parameter values. All in all, these results suggest that P&C is quite robust (or at least, not overly sensitive) to the choice of parameter values.

Some patterns also emerge from Tables 6, 7, and 8. The weighted *k*-core vertex scoring function systematically benefits from a large number of perturbed graphs ($M = 64$), while the other two functions have no clear preference. On all networks and for all scoring functions, deleting edges seems more beneficial than adding edges (almost always, $\varepsilon_d \geq \varepsilon_a$). For the unweighted *k*-core and PageRank functions, it also seems overall that selecting edges uniformly at random (with the Erdős-Rényi random graph model) when perturbing the network works better than selecting edges in a biased way (Chung-Lu model). It is the opposite for the weighted *k*-core function.

Finally, taking edges weights into account when perturbing the graph ($\delta_w = 1$), i.e., allowing edge weights to be incremented or decremented rather than purely and simply adding or deleting edges, works consistently better for unweighted *k*-core. This may seem surprising, since the unweighted *k*-core algorithm ignores edge weights. Still, when perturbing the original graph, it might be beneficial that the weakest edges (low weights) have higher likelihood of being deleted than the strongest edges (large weights), which $\delta_w = 1$ allows for. Conversely, since the PageRank and weighted *k*-core functions take edge weights into account, it might not be necessary to consider this information at perturbation time.

4.2 Word co-occurrence networks

It has been suggested that the keywords of a document are influential nodes within the word co-occurrence representation of that document, and that as such, keywords can be accurately identified by degeneracy-based techniques [38]. Therefore, in this section, we test whether perturbing and combining word co-occurrence networks improves keyword extraction performance.

Graph-based representations of text. There are several ways of representing text as a graph. Here, we use the classical statistical approach of [28, 39], based on the distributional hypothesis [14]. As shown in Figure 10, this method applies a fixed-size sliding window over the input text from start to finish. Each unique term in the document is represented by a node in an undirected graph, and two nodes are linked by an edge if the terms they represent co-occur within the window at least once. Edge weights indicate co-occurrence counts.

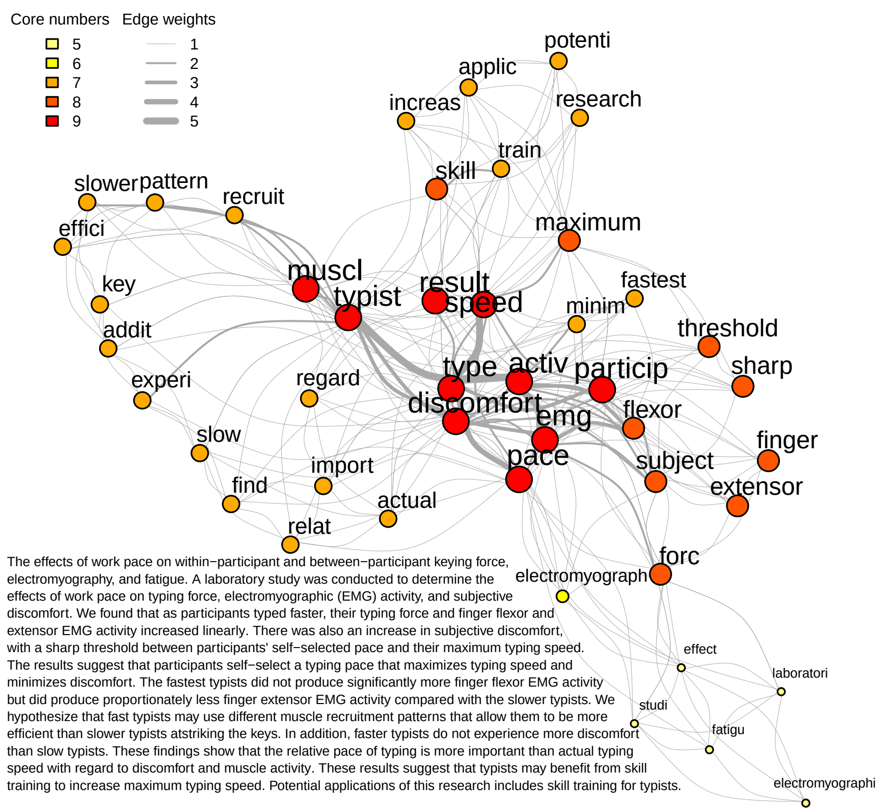

Fig. 10: Word co-occurrence network representation of document 1478 of the Hulth 2003 dataset. Only nouns and adjective are kept (and then stemmed). $W = 5$. The human keywords (stemmed) for this document are *work, pace, effect, emg, activ, subject, discomfort, finger, flexor, type, speed, typist, muscl, recruit, pattern, kei, forc, skill, train*.

Dataset. We used the well-known Hulth2003 dataset [17][1], which contains abstracts from the Inspec research article database. More precisely, we considered the standard set of 500 documents in the validation set and used the uncontrolled keywords assigned by human annotators as ground truth. The mean document size is 120 words and on average, 21 keywords (unigrams) are available for each document. Note that annotators were free to use any words to describe the documents, including ones that did not appear in the original text. Therefore, reaching perfect recall is impossible on this dataset.

Experimental setup. We represented each document in the validation set as a word co-occurrence network, using a window of size 5. As a pre-processing step, we applied part-of-speech tagging and retained only nouns and adjectives, following common practice [28]. Finally, we stemmed words with Porter's stemmer. Human keywords were also stemmed, but not filtered based on their part-of-speech tags (i.e., human keywords may contain verbs). The average number of nodes, edges, and diameter of the word networks were respectively 32, 155, and 3.6.

For unweighted and weighted k-core, we retained as keywords the words belonging to the main core of the network. Since our combination strategy is averaging, aggregated scores may be real numbers. Thus, we rounded them up to the nearest integer before extracting the main core. For PageRank, we extracted the top 33% nodes as keywords.

The following P&C parameters were grid searched: $\varepsilon_a : \{0, 0.1, 0.2, 0.3\}$, $\varepsilon_d : \{0, 0.1, 0.2, 0.3\}$, $N : \{8, 32, 96\}$, $\mathbb{G} : \{ER, CL\}$ and $\delta_w : \{0, 1\}$. Excluding the cases where $\varepsilon_a = \varepsilon_d = 0$, this made for 180 combinations.

Results. Performance is reported in terms of the usual precision, recall, and F1-score metrics in Table 10. Precision measures how many of the extracted keywords are found in the human keywords, while recall measures how many of the human keywords were extracted. The F1-score is the harmonic mean of precision and recall. Precision, recall and F1-score were computed for each document and averaged at the collection level (macro-averaging). The results for P&C are that obtained with the best parameter combination for each scoring function, shown in Table 11.

Like on social networks, using the aggregated scores in place of the original scores greatly improves performance for every node scoring function, with large absolute gains in F1-score ranging from 1.09 to 3.64. Even though looking at relative improvements is sufficient to show that our P&C strategy is effective, it

[1] https://github.com/snkim/AutomaticKeyphraseExtraction

is to be noted that the absolute scores we obtain are comparable to or exceed the state-of-the-art in unsupervised keyword extraction [38, 35].

Table 10: Precision, recall and F1-score comparison for keyword extraction on the Hulth2003 dataset. *cu*, *cw*, and *pr* respectively denote the unweighted k-core, weighted k-core and PageRank vertex scoring functions.

s	Scores	Precision	Recall	F1-score
cu	**aggregated**	**52.09**	**51.25**	**54.88**
	original	48.76	46.90	51.75
cw	**aggregated**	**50.53**	**48.54**	**52.50**
	original	48.07	46.81	48.86
pr	**aggregated**	**45.53**	**42.73**	**46.75**
	original	45.21	41.89	45.66

Interpretation. The fact that our P&C strategy improves performance even for PageRank is interesting. It tends again to indicate that either P&C works well even for stable vertex scoring functions, or that PageRank features some instability.

Also, as can be seen from Figure 11, most combinations of P&C parameters return aggregated scores that identify better keywords than the scores computed on the original graph. Like in the case of social networks, this is especially true for weighted k-core and PageRank. Overall, this suggests that P&C is quite robust to the choice of parameter values, and that with a better understanding of which parameters are important and what are good priors for them, the space to search can be reduced.

Indeed, it is clear not all parameters have a significant impact on final performance. For instance, looking at Table 11, we can see that the number of perturbed graphs M is not critical, while the edge addition and deletion probabilities ε_a and ε_d play an important role. More precisely, adding edges seems much more important than deleting edges, for all scoring functions. This is the opposite of the best configuration for social networks. Also, for unweighted and weighted k-core, the Erdős-Rényi (ER) random graph model, which selects edges uniformly, is clearly superior to the Chung-Lu (CL) model, where edges are more likely to be selected if their endpoints have high degrees. Conversely, the five best parameter combinations for PageRank always involve CL. One explanation could be that since PageRank is highly biased towards the hubs, it is more effective to focus on the edges incident on the hubs when perturbing the graph. However, CL was not the best for PageRank on the social networks, which tends to indicates that the choice of the perturbation model also depends on the graph.

We hypothesize that the optimal P&C parameter values depend jointly on the structure of the graph, on the vertex scoring function used, and on the end application. With more research, good priors can probably be obtained for different functions and graph signatures (size, density, diameter...), reducing the need for parameter tuning.

Application-wise, adding new edges or incrementing the weights of already existing edges in the case of word co-occurrence networks is equivalent to copying and pasting words from/to the input text, which can be seen as a form of *data augmentation*. It could also be interpreted as having a sliding window of stochastic size featuring an additional *masking* mechanism such that edges are drawn between a subset only of the words in each instantiation of the window. Augmenting the data and using sliding windows of stochastic sizes have been proved very beneficial in Computer Vision and NLP [23, 29], so this could explain why $\varepsilon_a \geq \varepsilon_d$ works well with word co-occurrence networks. More generally, perturbing graphs could also be seen as a form of *adversarial training* [41].

Table 11: Top 5 best P&C parameters for each scoring function (in terms of F1-score). *cu*, *cw*, and *pr* respectively denote unweighted *k*-core, weighted *k*-core, and PageRank.

δ_w	$\mathbb{G}$	M	ε_d	ε_a	**cu**	*cw*	*pr*
1	*ER*	8	0.1	0.3	**54.88**	50.67	45.99
0	*ER*	96	0.0	0.3	**54.85**	52.50	45.99
1	*ER*	96	0.3	0.3	**54.78**	51.76	45.65
0	*ER*	8	0.1	0.3	**54.69**	52.38	46.06
0	*ER*	32	0.0	0.3	**54.67**	52.31	45.90

δ_w	$\mathbb{G}$	M	ε_d	ε_a	*cu*	**cw**	*pr*
0	*ER*	96	0.0	0.3	54.85	**52.50**	45.99
0	*ER*	8	0.0	0.3	54.0.2	**52.38**	45.52
0	*ER*	8	0.1	0.3	54.69	**52.38**	46.06
0	*ER*	32	0.0	0.3	54.67	**52.31**	45.90
0	*ER*	96	0.1	0.3	54.62	**51.78**	46.20

δ_w	$\mathbb{G}$	M	ε_d	ε_a	*cu*	*cw*	**pr**
1	*CL*	32	0.0	0.3	50.94	45.33	**46.75**
0	*CL*	32	0.3	0.3	50.45	49.46	**46.57**
1	*CL*	8	0.0	0.3	51.86	46.99	**46.53**
0	*CL*	32	0.0	0.3	52.39	50.07	**46.47**
0	*CL*	32	0.2	0.3	52.22	50.49	**46.45**

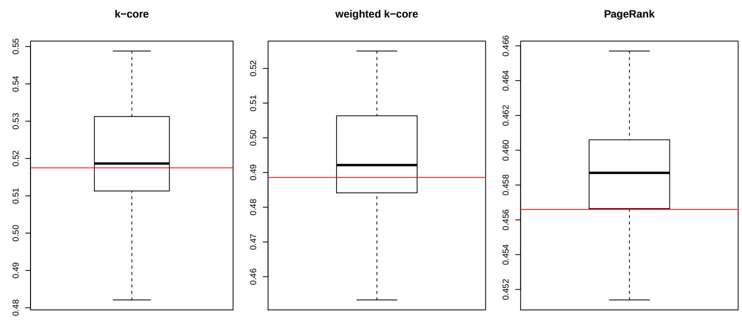

Fig. 11: Distribution of F1-score for the 180 combinations of P&C parameters. The F1-score obtained by extracting keywords based on their scores in the original graph is shown in red.

5 Theoretical analysis

The notation used in this section is summarized in Table 12.

Table 12: Summary of notation.

$G(V, E)$	undirected weighted graph, $n = \|V\|$
$v_i \in V$	the i-th vertex of G, $i \in \{1, \ldots, n\}$
G^*	true (unknown) graph underlying G
$G \sim G^*$	available graph (sample, snapshot of G^*)
$\tilde{G}_m \sim G$	m-th perturbed version of G
$\{\tilde{G}\}_{m=1}^{M} \equiv \{\tilde{G}\}$	set of M perturbed graphs
$s : V \mapsto \mathbb{R}^{\|V\|}$	vertex scoring function
$s_G(i)$	score of vertex v_i in G
$R = \{l(1), \ldots, l(n)\}$	true ranking on G, $l(i) > l(j) \Leftrightarrow s_G(i) > s_G(j)$
$\hat{R}$	ranking when s is applied on G, estimate of R
$\hat{R}_m$	ranking when s is applied on $\tilde{G}_m$; estimate of R
NDCG	goodness of $\hat{R}$ w.r.t. R

Setting. Let us assume the existence of a true but unavailable underlying graph G^*, of which the available graph G is a snapshot, or sample, so that G features the same nodes as G^* but has a slightly different edge set. This is analogous to the traditional assumption made in statistics and machine learning that *a given dataset represents a snapshot, or sample, from a true but unknown distribution*. Since G^* is unavailable, we have to find a way to emulate sampling from G^* by using only G. One solution is to perturb G.

True ranking. Let us also assume that there exists a true ranking $R = \{l(1), \ldots, l(n)\}$ of the nodes $\{v_1, \ldots, v_n\}$ of G, that associates each node v_i with one of K labels

$l(i)$, where v_i is ranked before v_j if $l(i) > l(j)$. $K \leq n$ as some nodes may have equivalent spreading capabilities. In the context of influential spreader detection, this true ranking can be computed like in subsection 4.1 by triggering a certain number of SIR epidemics from each node and averaging the number of nodes infected during the epidemic, the average counts serving as the node labels.

Objective. Let $s : V \mapsto \mathbb{R}^{|V|}$ be a vertex scoring function, i.e., a function that associates each node in G with a real number, and let $\hat{R}$ be the ranking induced by s on the nodes of G. $\hat{R}$ can be seen as an *estimate* of the true ranking R (hence the tilde notation), and s as an *estimator*. The objective of s is to estimate the true ranking R as well as possible.

Evaluation. Goodness of fit with respect to the true ranking R can be measured in terms of the normalized discounted cumulative gain (NDCG) [19], a widely-used metric for assessing the quality of rankings in information retrieval. The more good spreaders are placed on top of $\hat{R}$, the better the NDCG.

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}} \tag{2}$$

Where DCG is the discounted cumulative gain computed on $\hat{R}$ and IDCG is the ideal DCG computed on R. Therefore, NDCG is maximal and equal to 1 if $\hat{R}$ matches R exactly. Generally in information retrieval, the DCG is computed over a shortlist of the best results, but here, without loss of generality, we can assume that it is computed for the full list of n nodes:

$$\text{DCG} = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{3}$$

where i designates the rank of node v_i in the list. The relevance score rel_i of node v_i can be given by its label $l(i)$. As $\hat{R}$ is a random variable, NDCG can be considered a random variable by extension.

In each edge-perturbed version $\tilde{G}_m$ of G, the individual node scores, and by extension the rankings $\hat{R}_m$, randomly vary, as our perturbation strategy is stochastic. We can thus consider the $\hat{R}_m$ to be random variables. Moreover, since the $\tilde{G}_m$ are generated independently, the $\hat{R}_m$ are independent. Therefore, perturbing G is akin to *sampling independent realizations from the true underlying graph* G^*.

Definitions: bias and variance of a vertex scoring function.
We would like to study the impact of P&C on the quality of the estimate $\hat{R}$ of R. How well $\hat{R}$ fits R is indicated by the error, which traditionally in regression can be decomposed into bias and variance terms. We would therefore like to study the impact of P&C on the bias and variance of $\hat{R}$. Note that this is equivalent to studying the impact of P&C on the bias and variance of the ranking function s, since $\hat{R}$ is provided by s (the estimator). In what follows, we define the bias and variance of a vertex scoring function.

In the regression setting, $y = f(x) + \epsilon$, $\sigma^2 = \mathrm{var}[\epsilon]$, $\hat{f}$ is an estimator of f, and we have the following well-known breakdown of expected squared error of the estimation into (squared) bias, variance, and irreducible error terms:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \mathrm{bias}[\hat{f}(x)]^2 + \mathrm{var}[\hat{f}(x)] + \sigma^2 \tag{4}$$

$$\mathrm{bias}[\hat{f}(x)] = \mathbb{E}[\hat{f}(x) - f(x)] \tag{5}$$

$$\mathrm{var}[\hat{f}(x)] = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] \tag{6}$$

The expectation is computed for different samples drawn from the same underlying distribution. By analogy, in our setting, we can define the bias and variance of $\hat{R}$ as:

$$\mathrm{bias}[\hat{R}] = \mathbb{E}[1 - \mathrm{NDCG}] = 1 - \mathbb{E}[\mathrm{NDCG}] \tag{7}$$

$$\mathrm{var}[\hat{R}] = \mathbb{E}[(\mathrm{NDCG} - \mathbb{E}[\mathrm{NDCG}])^2] \tag{8}$$

The bias captures, on average, how close the estimated ranking $\hat{R}$ is to the true ranking R (which gets a NDCG of 1 per Equation 2), while the variance measures the instability of $\hat{R}$ (variability around its "mean"). The expectation is to be understood as computed over a set of observations of G^*.
Summing the squared bias term and the variance term gives us:

$$\mathrm{bias}^2[\hat{R}] + \mathrm{var}[\hat{R}] = \mathbb{E}[(\mathrm{NDCG} - 1)^2] \tag{9}$$

which can be interpreted as the expectation of the squared error like in the case of regression. The proof of Equation 9 is provided in the Appendix.

Definition: unstable vertex scoring function. A supervised learning algorithm is said to be unstable if small changes in its training set can cause large changes in its predictions (high variance). By analogy, we define a vertex scoring function s as unstable if it has high variance, i.e., if small changes in G can cause large changes in its induced ranking $\hat{R}$. Such changes include for instance addition and/or removal of a small fraction of the edges of G. Previous work provides evidence that, under this definition, the k-core decomposition can be considered unstable [1, 12]. Our experiments suggest that this might also be the case for PageRank (e.g., see Figure 4).

Perturbing and combining graphs reduces error. Recall that the aggregated score of node v_i for the node scoring function s is defined as the average of the scores its gets in each of the M perturbed graphs $\{\tilde{G}_1, \ldots, \tilde{G}_M\} \equiv \{\tilde{G}\}_{m=1}^{M}$ (or just $\{\tilde{G}\}$ for simplicity) generated from G:

$$s_{\mathrm{agg}}(i) = \frac{1}{M}\sum_{m=1}^{M} s_{\tilde{G}_m}(i) \tag{10}$$

where $s_{\tilde{G}_m}$(i) denotes the score of v_i in graph $\tilde{G}_m$. We can write:

$$\hat{R}_{\mathrm{agg}} = \mathbb{E}_{\{\tilde{G}\}}[\{\hat{R}_m\}_{m=1}^{M}] \tag{11}$$

which means that the aggregated estimate $\hat{R}_{\mathrm{agg}}$ of the true ranking R is the average of the individual estimates $\hat{R}_m$ over the M perturbed graphs. Similarly, the performance of the aggregated ranking, measured in terms of NDCG (itself a random variable, as previously explained), can be written:

$$\mathrm{NDCG}_{\mathrm{agg}} = \mathbb{E}_{\{\tilde{G}\}}[\mathrm{NDCG}(\{\tilde{G}\})] \tag{12}$$

Therefore, by developing the expectation of the squared error over $\{\tilde{G}\}$, given by Equation 9, and using Equation 12 above:

$$\begin{aligned} \mathbb{E}_{\{\tilde{G}\}}[(\mathrm{NDCG} - 1)^2] &= 1 - 2\mathbb{E}_{\{\tilde{G}\}}[\mathrm{NDCG}] + \mathbb{E}_{\{\tilde{G}\}}[\mathrm{NDCG}^2] \\ &= 1 - 2\mathrm{NDCG}_{\mathrm{agg}} + \mathbb{E}_{\{\tilde{G}\}}[\mathrm{NDCG}^2] \end{aligned} \tag{13}$$

Furthermore, since for any random variable X, $\mathbb{E}^2[X] \geq \mathbb{E}[X^2]$, using again Equation 12, and since $\mathbb{E}$ is monotone, we obtain:

$$\begin{aligned} \mathbb{E}_{\{\tilde{G}\}}[(\mathrm{NDCG} - 1)^2] &\geq 1 - 2\mathrm{NDCG}_{\mathrm{agg}} + \mathbb{E}^2_{\{\tilde{G}\}}[\mathrm{NDCG}] \\ &\geq (1 - \mathrm{NDCG}_{\mathrm{agg}})^2 \\ \mathbb{E}_{\{\tilde{G}\}}[(\mathrm{NDCG} - 1)^2] &\geq \mathbb{E}_{\{\tilde{G}\}}[(1 - \mathrm{NDCG}_{\mathrm{agg}})^2] \end{aligned} \tag{14}$$

Equation 14 suggests that ***the mean squared error of P&C (right-hand side) is always lower than or equal to the original mean squared error (left-hand***

side), which is an important result. The improvement can come from reducing bias, variance, or both.

Practical example. To quantify and understand the impact of P&C on bias and variance for a real sample (i.e., understand why Equation 14 holds in practice), we randomly selected 16 word co-occurrence networks from the Hulth2003 dataset, and generated 50 perturbed version of each network with $\varepsilon_a = 0.2, \varepsilon_d = 0.2, \mathbb{G} = ER$, and $\delta_w = 1$. As previously explained, this can be considered as drawing 50 independent realizations from the underlying graph that generated each network. With the unweighted *k*-core function, we then scored the nodes of each graph in the sample with and without using our P&C strategy[2], and computed the NDCG of each ranking (original and aggregated). We finally computed the sample bias and variance from the set of 50 NDCGs by using Equations 7 and 8. The true rankings had previously been calculated on the available graph via the SIR-based approach described in subsection 4.1. We repeated the same procedure for WikiVote, with $\varepsilon_a = \varepsilon_d = 0.05$ due to the much bigger size of the network.

Results are shown in Tables 13 and 14. As can be seen for word networks, P&C reduces both bias and variance, although its major contribution is in the consistent reduction of bias. On the WikiVote network, P&C reduces bias but increases variance.

Interpretation. The fact that the major contribution of our P&C strategy seems to stem from decreasing bias rather than variance suggests that our strategy differs from bootstrap aggregation (bagging). Indeed, bagging can increase variance when it fails (see e.g., [13]), but it is widely accepted that it cannot significantly reduce bias and that the error reduction is achieved almost entirely through reducing variance. Also, a bootstrap sample is of the same size as the original dataset, a property that our framework violates whenever $\varepsilon_a \neq \varepsilon_d$.

To further highlight how our approach differs from bagging, we reason by contradiction in what follows. If we were strictly emulating bagging, we would need to consider edges as observations. During perturbation, some edges would be selected more than once (their weights would be incremented), and some edges would not be selected at all. In the final perturbed network, 63.2% of unique edges would carry over from the original network, but no new edge would be present, as bootstrapping (i.e, sampling with replacement), does not involve creating new observations[3]. We would reduce error mainly by reducing variance as:

[2] with parameters $M = 16, \varepsilon_a = 0.1, \varepsilon_d = 0.3, \mathbb{G} = ER$, and $\delta_w = 0$.

[3] That would remove the need for ε_a and ε_d parameters (edges could only be removed or strengthened), at the cost of losing flexibility.

Table 13: Sample bias ($\times 10^2$) and variance ($\times 10^3$) of original and P&C scores computed on 50 (pseudo) observations of 16 randomly selected word co-occurrence networks from the Hulth2003 dataset, for the unweighted k-core scoring function. Bias and variance are computed as described in Equations 7 and 8 (the lower the better). Relative differences between Original and P&C are of interest here. P&C systematically reduces bias, but interestingly, increases variance in several cases. The average of the averages of the NDCGs is 57.52 for *original* and 72.66 for P&C (100 indicates a perfect ranking), which means that the rankings returned by P&C fit the true rankings much better.

	Original	P&C			Original	P&C	
bias	37.06	24.92	$\searrow$	bias	51.59	34.97	$\searrow$
var	4.78	1.17	$\searrow$	var	0.66	1.16	
bias	41.92	16.66	$\searrow$	bias	39.16	27.22	$\searrow$
var	12.83	4.72	$\searrow$	var	0.04	1.79	
bias	48.44	35.64	$\searrow$	bias	66.43	54.66	$\searrow$
var	6.02	5.43	$\searrow$	var	0.09	0.76	
bias	42.86	30.23	$\searrow$	bias	46.97	21.98	$\searrow$
var	0.08	4.44		var	1.54	2.71	
bias	45.16	31.23	$\searrow$	bias	26.40	20.75	$\searrow$
var	0.03	4.09		var	0.04	0.49	
bias	64.56	38.97	$\searrow$	bias	37.29	24.40	$\searrow$
var	1.87	4.04		var	0.27	2.17	
bias	24.75	14.05	$\searrow$	bias	33.76	18.68	$\searrow$
var	0.58	2.52		var	0.18	2.50	
bias	31.72	19.00	$\searrow$	bias	41.61	24.12	$\searrow$
var	0.58	6.78		var	6.85	6.60	$\searrow$

Table 14: Sample bias ($\times 10^2$) and variance ($\times 10^5$) of original and P&C scores computed on 48 (pseudo) observations of the WikiVote network, for the unweighted k-core scoring function. Bias and variance are computed as described in Equations 7 and 8 (the lower the better). Relative differences between Original and P&C are of interest here. P&C reduces bias, but not variance, which is initially very low. As indicated by $\overline{\text{NDCG}}$, the rankings returned by P&C fit the true ranking better.

	Original	P&C	
bias	81.08	79.68	$\searrow$
var	0.00	0.64	
$\overline{\text{NDCG}}$	18.92	20.32	

$$
\begin{aligned}
\operatorname{var}[\hat{R}_{\text{agg}}] &= \operatorname{var}\left[\frac{1}{M}\sum_{m=1}^{M}\hat{R}_m\right] \\
&= \frac{1}{M^2}\sum_{m=1}^{M}\operatorname{var}[\hat{R}_m]
\end{aligned}
\tag{15}
$$

the extent to which error would be reduced would thus depend on the amount of uncorrelation among the rankings $\hat{R}_m$, as correlation adds positive covariance terms to the right-hand side of Equation 15. Without diversity, Equation 14 collapses to an equality. Therefore, if our approach was equivalent to bagging, it would only work for unstable vertex scoring functions. However, it works well in practice even for PageRank, which is usually considered to be quite robust to link-based perturbations [18, 31] (although we noticed in our experiments that PageRank features some instability, see Figure 2). Furthermore, we observe our method to significantly reduce bias and increase variance in practice (see Tables 13 and 14). For all these reasons, our P&C strategy for networks clearly deviates from bootstrap aggregation.

Rather, we think that our approach is more closely related to *noise injection* (*noisy* and *smoothed* bootstrap approaches in [34, 37]) and *data augmentation* strategies.

6 Conclusion

We proposed what is, to the best of our knowledge, the first application of the Perturb and Combine (P&C) strategy, well-known in Machine Learning, to networks. Our meta-algorithm generates a number of perturbed versions of a given graph, separately applies a vertex scoring function to each graph, and combines back the results into aggregated scores.

Experiments conducted on real-world networks of different types and various sizes provide evidence that the aggregated scores returned by our P&C procedure allow to identify significantly better spreaders than using the scores in the original graphs. Furthermore, even though our approach was motivated by unstable vertex scoring functions, a bias-variance analysis suggests that our method primarily reduces bias. Therefore, it can in theory be effective for all scoring functions, not only unstable ones.

From a computational perspective, our proposed P&C procedure is trivially parallelizable. Computing the aggregated scores thus does not require much more time than scoring the original network only, and P&C can thus be used in practice.

Finally, we think that P&C could improve performance of existing methods for other graph mining tasks, like community detection or densest subgraph finding.

References

1. Adiga, A., Vullikanti, A.K.S.: How robust is the core of a network? In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 541–556. Springer (2013)
2. Balog, K., Azzopardi, L., De Rijke, M.: Formal models for expert finding in enterprise corpora. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 43–50. ACM (2006)
3. Batagelj, V., Zaveršnik, M.: Generalized cores. arXiv preprint cs/0202039 (2002)
4. Breiman, L.: Bagging predictors. Machine learning **24**(2), 123–140 (1996)
5. Breiman, L.: Bias, variance, and arcing classifiers (1996)
6. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
7. Chakrabarti, D., Wang, Y., Wang, C., Leskovec, J., Faloutsos, C.: Epidemic thresholds in real networks. ACM Transactions on Information and System Security (TISSEC) **10**(4), 1:1–1:26 (2008)
8. Chung, F., Lu, L.: The average distances in random graphs with given expected degrees. Proceedings of the National Academy of Sciences **99**(25), 15879–15882 (2002)
9. Dorogovtsev, S.N., Mendes, J.F.: Evolution of networks: From biological nets to the Internet and WWW. OUP Oxford (2013)
10. Erdös, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hung. Acad. Sci **5**, 17–61 (1960)
11. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: SIGCOMM '99: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. pp. 251–262 (1999)
12. Goltsev, A.V., Dorogovtsev, S.N., Mendes, J.F.F.: k-core (bootstrap) percolation on complex networks: Critical phenomena and nonlocal effects. Physical Review E **73**(5), 056101 (2006)
13. Grandvalet, Y.: Bagging equalizes influence. Machine Learning **55**, 251–270 (2004)
14. Harris, Z.S.: Distributional structure. Word **10**(2-3), 146–162 (1954)
15. Ho, T.K.: The random subspace method for constructing decision forests. IEEE transactions on pattern analysis and machine intelligence **20**(8), 832–844 (1998)
16. Hoppensteadt, F.: Mathematical Theories of Populations: Deomgraphics, Genetics, and Epidemics, vol. 20. Siam (1975)
17. Hulth, A.: Improved automatic keyword extraction given more linguistic knowledge. In: Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 216–223. Association for Computational Linguistics (2003)
18. Ipsen, I.C., Wills, R.S.: Mathematical properties and analysis of googleâĂŹs pagerank
19. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Transactions on Information Systems (TOIS) **20**(4), 422–446 (2002)
20. Kermack, W.O., McKendrick, A.G.: Contributions to the mathematical theory of epidemics. ii. the problem of endemicity. vol. 138, pp. 55–83. JSTOR (1932)
21. Kitsak, M., Gallos, L.K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H.E., Makse, H.A.: Identification of influential spreaders in complex networks. Nature physics **6**(11), 888–893 (2010)

22. Klimt, B., Yang, Y.: The enron corpus: A new dataset for email classification research. ECML '04: Proceedings of the 15th European Conference on Machine Learning pp. 217–226 (2004)
23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
24. Leskovec, J., Adamic, L.A., Huberman, B.A.: The dynamics of viral marketing. ACM Transactions on the Web (TWEB) **1**(1), 5 (2007)
25. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
26. Lusseau, D.: The emergent properties of a dolphin social network. Proceedings of the Royal Society of London B: Biological Sciences **270**(Suppl 2), S186–S188 (2003)
27. Malliaros, F.D., Rossi, M.E.G., Vazirgiannis, M.: Locating influential nodes in complex networks. Scientific reports **6**, 19307 (2016)
28. Mihalcea, R., Tarau, P.: TextRank: bringing order into texts. In: Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics (2004)
29. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
30. Motter, A.E., Lai, Y.C.: Cascade-based attacks on complex networks. Phys Rev E **66**, 065102 (2002)
31. Ng, A.Y., Zheng, A.X., Jordan, M.I.: Link analysis, eigenvectors and stability. In: International Joint Conference on Artificial Intelligence. vol. 17, pp. 903–910. LAWRENCE ERLBAUM ASSOCIATES LTD (2001)
32. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)
33. Pastor-Satorras, R., Vespignani, A.: Epidemic spreading in scale-free networks. Phys. Rev. Lett. **86**, 3200–3203 (2001)
34. Raviv, Y., Intrator, N.: Bootstrapping with noise: An effective regularization technique. Connection Science **8**(3-4), 355–372 (1996)
35. Rousseau, F., Vazirgiannis, M.: Main core retention on graph-of-words for single-document keyword extraction. In: European Conference on Information Retrieval. pp. 382–393. Springer (2015)
36. Seidman, S.B.: Network structure and minimum degree. Social networks **5**(3), 269–287 (1983)
37. Silverman, B.W.: Density estimation for statistics and data analysis. Routledge (1986)
38. Tixier, A., Malliaros, F., Vazirgiannis, M.: A graph degeneracy-based approach to keyword extraction. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 1860–1870 (2016)
39. Tixier, A., Skianis, K., Vazirgiannis, M.: Gowvis: a web application for graph-of-words-based text visualization and summarization. Proceedings of ACL-2016 System Demonstrations pp. 151–156 (2016)
40. Zeng, D., Chen, H., Lusch, R., Li, S.H.: Social media analytics and intelligence. IEEE Intelligent Systems **25**(6), 13–16 (2010)
41. Zügner, D., Akbarnejad, A., Günnemann, S.: Adversarial attacks on neural networks for graph data. arXiv preprint arXiv:1805.07984 (2018)

Appendix: Proof of Equation 9

We start by summing the squared bias and the variance (bias and variance have been defined in Equations 7 and 8).

$$\mathrm{bias}^2[\hat{R}] + \mathrm{var}[\hat{R}] = (1 - \mathbb{E}[\mathrm{NDCG}])^2 + \mathbb{E}[(\mathrm{NDCG} - \mathbb{E}[\mathrm{NDCG}])^2] \tag{16}$$

Developing the left term of the right-hand side gives:

$$\mathrm{LRHS} = 1 - 2\mathbb{E}[\mathrm{NDCG}] + \mathbb{E}^2[\mathrm{NDCG}] \tag{17}$$

Since the expected value of a sum of random variables is equal to the sum of their expected values, the expected value of a constant is equal to that constant, and for all random variable X and $\lambda \in \mathbb{R}$, $\mathbb{E}[\lambda X] = \lambda\mathbb{E}[X]$, developing the right term of the right-hand side gives:

$$\begin{aligned} \mathrm{RRHS} &= \mathbb{E}[\mathrm{NDCG}^2 - 2\mathbb{E}[\mathrm{NDCG}]\mathrm{NDCG} + \mathbb{E}^2[\mathrm{NDCG}]] \quad (18) \\ &= \mathbb{E}[\mathrm{NDCG}^2] - 2\mathbb{E}^2[\mathrm{NDCG}] + \mathbb{E}^2[\mathrm{NDCG}] \quad (19) \end{aligned}$$

We thus get:

$$\begin{aligned} \mathrm{bias}^2[\hat{R}] + \mathrm{var}[\hat{R}] &= \mathrm{LRHS} + \mathrm{RRHS} \quad (20) \\ &= 1 - 2\mathbb{E}[\mathrm{NDCG}] + \mathbb{E}[\mathrm{NDCG}^2] \quad (21) \\ &= \mathbb{E}[(\mathrm{NDCG} - 1)^2] \quad (22) \end{aligned}$$

Which gives us Equation 9.