

Refutations on “Debunking the Myths of Influence Maximization: An In-Depth Benchmarking Study”

Wei Lu
Rupert Labs
465 Fairchild Drive
Mountain View, CA 94043
w.lu@alumni.ubc.ca

Xiaokui Xiao
Nanyang Technological Univ.
50 Nanyang Avenue
Singapore 639798
xkxiao@ntu.edu.sg

Amit Goyal
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
goyalamit@google.com

Keke Huang
Nanyang Technological Univ.
50 Nanyang Avenue
Singapore 639798
khuang005@ntu.edu.sg

Laks V.S. Lakshmanan
Univ. of British Columbia
201-2366 Main Mall
Vancouver, B.C., Canada
laks@cs.ubc.ca

ABSTRACT

In a recent SIGMOD paper titled “*Debunking the Myths of Influence Maximization: An In-Depth Benchmarking Study*”, Arora et al. [1] undertake a performance benchmarking study of several well-known algorithms for influence maximization. In the process, they contradict several published results, and claim to have unearthed and debunked several “myths” that existed around the research of influence maximization.

It is the goal of this article to examine their claims objectively and critically, and refute the erroneous ones. Our investigation discovers that first, the overall experimental methodology in Arora et al. [1] is flawed and leads to scientifically incorrect conclusions. Second, the paper is riddled with issues specific to a variety of influence maximization algorithms, including buggy experiments, and draws many misleading conclusions regarding those algorithms. Importantly, they fail to recognize the trade-off between running time and solution quality, and did not incorporate it correctly in their experimental methodology. In this article, we systematically point out the issues present in [1] and refute 11 of their misclaims.

1. INTRODUCTION

Influence maximization in social networks is a well-studied problem with applications in viral marketing, the study of propagation of infections and innovation, and community detection, to name a few. Over the last decade and a half, substantial amount of research has been conducted on this problem, resulting in a plethora of algorithms ranging from heuristics to approximation algorithms, with varying levels of quality, running time efficiency, and memory consumption. In a recent paper titled “*Debunking the Myths of Influence Maximization: An In-Depth Benchmarking Study*” (SIGMOD 2017), Arora et al. [1] undertake a performance benchmarking study of several well-known algorithms for influence maximization. In the process, they claim to unearth several “myths” which they then claim to debunk. In this article, we examine their claims objectively and criti-

cally, point out the errors in many of their claims and systematically refute them. We do this by trying to reproduce their claimed experimental results. Importantly, our analysis shows that first, the overall experimental methodology employed by Arora et al. [1] is flawed and leads to scientifically incorrect conclusions. Second, a series of experimental results claimed in [1] are a direct consequence of improper data preparation. We directly refute these claims by running those experiments with proper dataset preparation, and by running independent experiments in two different institutions. Our findings squarely contradict their claims. Third, we identify misleading claims made in [1] that fail to properly take into account the trade-off between running time and memory consumption on one hand and accuracy and quality of the solution obtained on the other. We illustrate with examples that these misleading claims can be used to draw obviously incorrect conclusions.

1.1 Influence Maximization Recap

We begin with a quick review of the problem. Influence Maximization (IM) is an optimization problem studied extensively in the social network data mining literature during the past decade and a half. Originally motivated by viral marketing [11, 25], IM was first formulated as a discrete optimization problem in a seminal paper by Kempe et al. [19]. We are given (1) a directed graph $G = (V, E, p)$ as a social network, where nodes are individuals, edges represent relationships, and $p : E \rightarrow [0, 1]$ specifies pairwise influence probabilities (or weights) between nodes; (2) a positive integer k , and (3) a stochastic diffusion model M which specifies probabilistic rules on how influence propagates from one node to another in the graph. In such a network, activating a set of nodes $S \subseteq V$ leads to a cascade of actions in which the nodes S , typically called *seed nodes*, activate at time $t = 0$, and activation propagates between nodes in discrete time steps according to the model M . Two popular diffusion models in the literature are *independent cascades* (IC) and *linear threshold* (LT). We refer the reader to [19] for their description. Since M is stochastic, the number of nodes that activate in a cascade is a random number. The *expected spread* of S is defined as the expected total number of activated nodes in a random cascade started by seed nodes S . Formally, the expected spread is a function $\sigma_M : 2^V \rightarrow \mathbb{R}$

that maps every set of nodes to a positive real number, representing the expected number of activated nodes over all possible cascades started by seed nodes S .

The optimization problem is to find a set $S \subseteq V$ of k seeds, such that by activating S the expected spread of S under model M , denoted $\sigma_M(S)$ (or just $\sigma(S)$ when M is understood), is maximized. IM is a computationally challenging problem: (i) it is NP-hard under a large family of stochastic diffusion models [19] and (ii) for a given set of nodes S , computing the spread function $\sigma(S)$ is #P-hard under both IC and LT models [5, 7]. In response to this two-fold challenge, there has been a large body of research on efficient and scalable IM algorithms, e.g., [2, 5–7, 10, 16–18, 20, 21, 23, 26, 27]. For convenience, technical descriptions of the algorithms relevant to this article are provided in Appendix A.

In their paper, Arora et al. [1] undertake a performance benchmarking study of several well-known algorithms for influence maximization, including [5, 7, 16–18, 21, 26, 27]. In this work, we critically examine the experimental methodology used in [1] and their various claims against the IM algorithms proposed in the papers cited above.

1.2 Overview of Flaws and Misclaims in Arora et al. [1]

Flaws in Experimental Design. One of the most emphasized aspects in Arora et al. [1] is the running time comparison. To this end, they focus on the following question ([1], Section 5.1.1): “How much time it takes for an algorithm to reach to *its* near-optimal spread?” (emphasis added by us). They define “near-optimal” in an empirical sense. We review their definition in Section 2, where we also conduct an in-depth analysis. Suffice it for now to note that the “near-optimal” spread is algorithm specific and can thus be drastically different for different algorithms. Thus, any comparison of the running times of different IM algorithms based on such an algorithm specific “near-optimal” spread necessarily holds the algorithms to different bars! To appreciate the seriousness of this issue, consider the following example.

EXAMPLE 1. *Consider two hypothetical IM algorithms \mathcal{A} and \mathcal{B} . Suppose that \mathcal{A} is an approximation algorithm that employs sophisticated sampling techniques, and thus the more samples it draws, the higher the quality of seed sets it can achieve. Suppose the “near-optimal quality” of \mathcal{A} corresponds to an estimated expected spread of 1000 for $k = 10$ seeds on some dataset D , and for that it takes 10 minutes. Algorithm \mathcal{B} is a simple heuristic that takes 1 minute to get to its “near-optimal quality”, namely a spread of 100 for $k = 10$ seeds on the same dataset D . Suppose also that with fewer samples, Algorithm \mathcal{A} achieves an estimated expected spread of 100 and it takes only 0.1 minute to achieve this, again with 10 seeds. A comparison between Algorithms \mathcal{A} and \mathcal{B} solely based on the time taken by each to achieve its own “near-optimal” spread would conclude that \mathcal{B} is a better algorithm despite the fact that \mathcal{A} achieves what \mathcal{B} achieves, an order of magnitude faster!*

In addition to the problematic issue of holding different algorithms to different bars, as demonstrated in Example 1, there are two more problems associated with the experimental methodology of [1], as discussed in Section 2: (i) As it turns out, the definition of “near-optimal” spread in [1] allows for a solution that can be *arbitrarily* worse than the

optimal spread, which calls into question the legitimacy of the term “near-optimal” spread; (ii) No clear description is offered by Arora et al. [1] on how exactly the parameters of the algorithms were set in order to obtain the “near-optimal” spread; as a result, it is unclear how each IM algorithm was allowed to run to reach such spread.

Measuring Standard Deviation. Apart from the flawed experimental design, other fundamental issues exist in their approach. In Figure 12 of [1], Arora et al. reported a set of standard deviation values, which are in turn used to determine the “near-optimal quality” of an algorithm. By design, the correctness of these standard deviation measurements is critical to the correctness of their entire set of running time experiments. However, in our attempts to reproduce their Figure 12, we found a *significant discrepancy of 10x: The standard deviation values obtained by us (validated at both NTU and UBC independently) are 10 times larger than the values claimed in Figure 12 of [1]*. Section 2 will explain how this issue has serious implications for the validity of all other experiments in [1]¹.

Algorithm-Specific Major Issues. Arora et al. [1] claimed to unearth several “myths” and criticized a variety of IM algorithms in the process of “debunking” those “myths”. Unfortunately, many of such claims are false. We shall refute them in detail in Section 3. Here is an overview of those claims and our refutation.

- TIM and TIM⁺ are the first scalable approximation algorithms for IM proposed by Tang et al. [27] based on the notion of reverse-reachable sets [2]. In [26], Tang et al. proposed an improvement to TIM/TIM⁺, called IMM, which leverages martingale theory to derive much tighter bounds on the number of samples required for a given guaranteed accuracy ϵ of spread estimation, compared to TIM/TIM⁺. IMM is much more scalable than TIM⁺ in the sense that for any given accuracy guarantee ϵ , IMM requires far fewer samples to achieve that guarantee than TIM⁺.

Arora et al. [1] ignore the notion of theoretical accuracy guarantee and instead compare TIM⁺ and IMM on *empirical accuracy* and reach an erroneous conclusion that TIM⁺ sometimes scales better than IMM. This reveals a lack of understanding of what theoretical guarantees are about, which are concerned with the worst case, as opposed to empirical accuracy. In Section 3.1, we provide more arguments and details refuting their conclusion. To help appreciate the flaw in their scaling argument, we provide a simple example in Section 3.1. The example follows the same argument as used by [1] and leads to the conclusion that Chebyshev’s inequality is better than the Chernoff bound, which is clearly wrong!

- Arora et al. claim that the SimPath algorithm [17] failed to finish on two datasets after 2400 hours (100 days!). A careful examination of their results and the datasets used reveals that they did not in fact preprocess the datasets correctly before running the SimPath code (released in

¹In our email correspondences with them, the authors of [1] stated that they performed “binning” and “smoothing” to remove outliers before calculating standard deviations. We reached out to them again to request the relevant source code for performing the exact same binning and smoothing so that we could reproduce their results. Despite multiple requests, we are still waiting for their source code.

[17]), and as a result were stuck in an infinite loop. Without doing due diligence, they apparently simply let the code run for 100 days and concluded it does not finish in 100 days. Our experiments confirm that with correctly prepared data, SimPath takes only 8.6 and 667 minutes respectively to select 200 seeds on the two datasets tested, i.e., DBLP and YouTube (Section 3.2). One of the consequences of the incorrect results above is their claim that between the algorithms SimPath and LDAG, LDAG is more robust and is significantly faster under “uniform” LT model. As we discuss in Section 3.2, our experiments contradict their claims.

Issues with Overall Recommendations by [1]. In their paper, Arora et al. make a recommendation for which IM algorithm to use under what circumstances. This is expressed in the form of a decision tree (Figure 11(b) in [1]). As a preview of the problems associated with this recommendation, notice that the decision tree recommends that whenever one has a low memory machine, the IM algorithm of choice is EasyIM [13]. We note that the EaSyIM algorithm was proposed by Galhotra, Arora, and Roy in SIGMOD 2016 [13]. Both Galhotra and Arora are authors of [1] as well. The rationale offered by them is that EasyIM is supposedly the most memory efficient IM algorithm and so should be used when available memory is limited. Indeed, as the authors point out, EasyIM stores only a scalar per node of the network, which requires only a fraction of the memory needed to store, e.g., reverse reachable sets. By this argument, the *Random* algorithm, which randomly selects k nodes from the network as seeds, requires even less memory since it does not need to store anything! However, it is well known that on many datasets, the expected spread achieved by the *Random* algorithm is quite poor. What is missing in such a recommendation is a consideration of the trade-off between memory usage and spread achieved. We elaborate on the issues and misleading claims on EaSyIM [13] in Section 4.

1.3 Roadmap

The rest of our article is organized as follows. The background knowledge on IM as well as the algorithms covered in this article are covered in Appendix A. Section 2 analyzes the experimental design and methodology of [1] in detail and points out the flaws therein. Section 3 focuses on refuting algorithm-specific mis-claims. Section 4 discusses various other issues, including the problems associated with their claims on EaSyIM [13]. Appendix B discusses the issue of CELF vs CELF++, while Appendix C briefly remarks on the importance of understanding the source code of the algorithms that one undertakes to benchmark.

2. ISSUES IN EXPERIMENTAL METHODOLOGY AND SETTINGS

2.1 Standards of Benchmarking

There exist several different metrics for evaluating an IM algorithm. Recall that some algorithms are just heuristics and lack guarantees on the expected spread of the seed set they return. We can only calibrate them on the empirical spread observed. For approximation algorithms, there is both empirical spread and the theoretical *guarantee* on the worst-case spread. It is important to recognize the difference between these two. Besides spread, there are factors

such as running time and memory usage. Any evaluation and comparison of IM algorithms should take into account the trade-off between resources like running time and memory on one hand and the achievable spread, i.e., empirical or guaranteed spread. Achieving higher spread usually requires larger computation costs in terms of running time and memory usage.

Algorithms often have one or more parameters to allow users to control this tradeoff, e.g., to reduce the computation time of the algorithm at the cost of spread, or vice versa. For example, in case of Greedy with MC simulations, the #MC simulations is a parameter that controls the trade-off between accuracy and running time. In case of TIM⁺ and IMM, the accuracy parameter ϵ can be used to trade accuracy for memory as well as running time. As a consequence, the setting of such parameters is crucial for benchmarking different algorithms carefully. A standard practice for parameter setting is to tune the parameters to *ensure that all algorithms are held to the same performance bar on one or more metrics, and then gauge their performance on the other metrics.*

Suppose that for some reason, we wish to deviate from the standard practice and choose an alternative methodology for benchmarking IM algorithms, then we must ensure that the alternative methodology satisfies some minimum requirements for soundness. Consider Figure 1 for an example. It illustrates the expected spreads of two hypothetical IM algorithms when their running time varies (due to parameter tuning), for a given dataset and a seed set size $k = 200$. Observe that, in terms of the tradeoff between spread achieved and computation cost, Algorithm \mathcal{A} completely dominates Algorithm \mathcal{B} , since the former (i) yields a higher expected spread when the two algorithms have the same computation time, and (ii) incurs a smaller running time when the two algorithms offer the same expected spread. As such, *if a methodology for evaluating IM algorithms is sound, then it should not conclude that Algorithm \mathcal{B} is more favorable than Algorithm \mathcal{A} in terms of running time or spread achieved when $k = 200$.* Unfortunately, this is the very conclusion that the methodology of [1] would draw, and hence it does not pass this soundness check, as we elaborate below.

2.2 Flaws in Arora et al.’s Methodology

Essentially, the experimental methodology of [1] is driven by the following question: *How much time it takes for an algorithm to reach to its near optimal spread?* To answer this question, Arora et al. choose a *fixed* parameter setting for each influence maximization algorithm A , and then evaluate A on three metrics: spread achieved, computation time, and memory consumption. Specifically, if algorithm A has a parameter p , then they first run A on four small datasets, with a fixed seed set size k ($k=200$ in their settings), and with p varying in a certain range. Let S_D be the set of seed sets returned by A on a dataset D for different p . Arora et al. examine the expected spread of each seed set in S_D using 10000 rounds of Monte-Carlo simulations, and identify the seed set S_D^* whose expected spread μ_D^* is the largest. In addition, they measure the standard deviation sd_D^* of the spread of S_D^* in the above 10000 Monte-Carlo rounds. The final parameter value p is set to a value that minimizes the computation of A , subject to the constraint that A should achieve an expected spread at least $\mu_D^* - sd_D^*$, across all four small datasets. In other words, they set p in a manner such

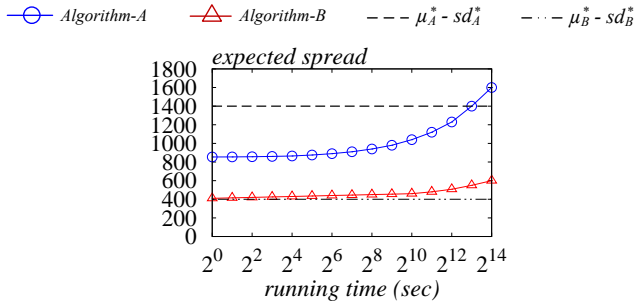


Figure 1: The tradeoff between expected spread and running time for two conceptual algorithms on a certain dataset, with seed set size $k = 200$.

that A 's spread achieved is close to *its* “best” (or, “near-optimal”). It is worth noting that once p is chosen for an algorithm, *it remains the same across all datasets, including the bigger ones.*

2.2.1 Issue 1: Different and arbitrary bars for different algorithms

Simply put, the “best” expected spread is algorithm specific and can be considerably different. *By construction, then different algorithms are not held to the same bar w.r.t. expected spread.* If an algorithm’s best expected spread is higher than another, then Arora et al.’s methodology would require the former to achieve a higher spread than the latter when it evaluates the two algorithms’ running time and memory consumption. Such a setting unfairly penalizes algorithms whose best expected spreads are large, and could lead to erroneous conclusions regarding their efficiency and memory overheads, as well as wrong recommendations for algorithms to use for given situations. In other words, the fact that an algorithm is capable of achieving a higher quality of seed sets will be used against it in running time comparisons!

In the context of Figure 1, where $k = 200$, Algorithm A (resp. Algorithm B) achieves its largest expected spread $\mu_A^* = 1600$ (resp. $\mu_B^* = 600$) when its running time equals 2^{14} seconds. Assuming that $sd_A^* = sd_B^* = 200$, then Arora et al.’s setting would require Algorithm A to yield an expected spread of at least $\mu_A^* - sd_A^* = 1400$, in which case its running time is at least 2^{13} seconds. In contrast, Algorithm B is only required to achieve an expected spread of at least $\mu_B^* - sd_B^* = 400$, in which case its computation time can be as small as 1 second. Evaluating the two algorithms’ efficiency under such a setting is clearly unfair to Algorithm A , and could lead to *incorrect* conclusions such as the following:

1. Algorithm A is not scalable due to its excessive computation time needed to achieve a spread of 1400, even though this is more than 3x the spread 400 expected of Algorithm B .
2. Algorithm B is more efficient than Algorithm A , despite the fact that Algorithm A can achieve the same spread of 400 in lower running time!

Both conclusions clearly contradict the fact that Algorithm A dominates Algorithm B , as depicted in Figure 1.

Now take a concrete example: In Section 5.3.1 of [1], Arora et al. conclude that IMM is not scalable under the IC model. However, this conclusion is drawn solely based on IMM’s running time when it is required to achieve an extremely

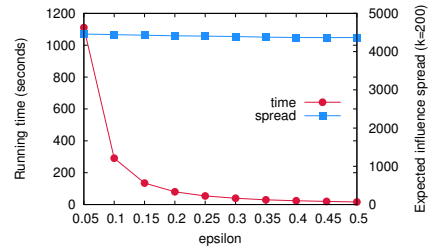


Figure 2: IMM on HepPH-IC: Running time (left Y-axis) and spread at $k = 200$ (right Y-axis), with varying ϵ .

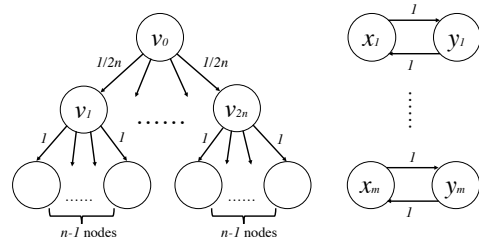


Figure 3: A counter example showing $\mu^* - sd^*$ giving arbitrarily bad spread. In the example graph, we let $m = n^3$. (Credit: Yang and Pei [29])

high accuracy (with its parameter ϵ set to 0.05), as Arora et al. experimental methodology demands that IMM should yield an expected spread close to *its* best. Meanwhile, other methods (e.g., EaSyIM) are not held to the same bar of spread, since their best expected spreads are lower than that of IMM. If we allow IMM to output less accurate results as other methods do, then it would become much more efficient and scalable.

For instance, consider Figure 2. It depicts results of our experiments on IMM’s running time and spread achieved (at $k = 200$) on HepPH dataset, under the IC model, with ϵ varied over 0.05, 0.1, \dots , 0.5. Clearly, as ϵ increases, *the drop of spread is barely visible, while the running time goes down drastically.* We can see that $\epsilon = 0.05$, the choice in [1], is almost an extreme adversarial setting for IMM: It took significantly more time, but produced only marginally better spread than larger ϵ ’s. For instance, *from $\epsilon = 0.05$ to 0.5, the running time speed-up is about 68x, while the drop in spread is only 2.1%!* Similar observations hold for other datasets too. It follows that without sacrificing spread too much, it is possible to set ϵ to higher values, thus *easily* achieving scalability.

Hence, Arora et al.’s claim that IMM does not scale is invalid, and it is a consequence of their flawed experimental methodology.

2.2.2 Issue 2: How close to optimal is “near-optimal” spread?

Arora et al. [1] define the “near-optimal” influence spread using $\mu^* - sd^*$, as mentioned in the beginning of Section 2. This, unfortunately, is an ad-hoc choice and we show through a counterexample contributed by Yu Yang and Jian Pei [29] that the value of $\mu^* - sd^*$ can be *arbitrarily* worse than the optimal influence spread. Consider Figure 3, where the graph is constructed as follows. Let $n \geq 2$ be a positive integer. There is a node v_0 that has $2n$ outgoing edges,

pointing to v_1, v_2, \dots, v_{2n} each with an influence probability of $\frac{1}{2n}$. Each v_i ($i \in [1, 2n]$) has $n-1$ out-neighbors each with influence probability 1. Then we create $m = n^3$ 2-cliques with influence probabilities being 1 on both edges.

Let $I(S)$ denote the number of active nodes by the end of a propagation when S is the seed set. Clearly, by definition, $\mathbb{E}[I(S)] = \sigma(S)$. For the example graph in Figure 3, under the independent cascade model, when $k = 1$, it can be verified that the following holds:

$$\mu := \mathbb{E}[I(\{v_0\})] = \sigma(\{v_0\}) = 1 + n, \quad (1a)$$

$$sd := \sqrt{\text{Var}[I(\{v_0\})]} = n\sqrt{1 - \frac{1}{2n}}, \quad (1b)$$

$$\mu - sd = 1 + n \left(1 - \sqrt{1 - \frac{1}{2n}}\right) \leq 1 + n\frac{1}{2n} = 1.5. \quad (1c)$$

On the other hand, for all $2n^3$ nodes in the 2-cliques structure, their expected spread is $2 > 1.5 = \mu - sd$. If we randomly select a node from this graph, then w.h.p. we will end up with a node of spread 2. Arora et al. [1] use $\mu^* - sd^*$ as the threshold for “near-optimal” spread. Since μ^* and sd^* are empirical estimates of μ and sd , one of the following scenarios must hold true if their approach is followed:

1. μ^* and sd^* are reasonably close to μ and sd respectively. Thus, $\mu^* - sd^*$ should be reasonably close to $\mu - sd$ which is bounded by a small constant 1.5 (Eq (1c)). As a result, the threshold $\mu^* - sd^*$ used for defining “near-optimal” spread can be arbitrarily smaller than the true optimal spread μ since it has a $\Omega(n)$ gap to the latter. This invalidates the use of $\mu^* - sd^*$ as the threshold for defining “near-optimal” spread.
2. The quantity $\mu^* - sd^*$ is reasonably close to the optimal spread μ , thus justifying a definition of “near-optimal” based on the threshold $\mu^* - sd^*$. However, in this case, one or more of the quantities in $\{\mu^*, sd^*\}$ must be a poor estimate of their counterparts in $\{\mu, sd\}$. This, by definition, calls into question the accuracy of μ^* and/or sd^* itself, and thus the validity of all experimental results based on such poor estimates will be questionable at best.

2.2.3 Issue 3: Notion of reasonable time limit is arbitrary and unclear

In addition to the fundamental issues mentioned above, Arora et al.’s methodology suffers from yet another critical issue. Recall that while choosing the value of parameter p (which is used to control the trade-off between running time and expected spread), Arora et al. allow algorithms to run for a “reasonable time limit” (Section 5.1.1 of [1]). However, unfortunately, no definition or value of reasonable time limit for any of the studied algorithms is provided. This makes the notion arbitrary and unclear, and worse, leaves “reasonable time limit” open to interpretation. Clearly, as more time is allowed, the parameter p gets stricter: e.g., more MC simulations or smaller accuracy parameter ϵ . In a fair world, a strict upper bound T on running time would have been provided, for *all* algorithms, making reasonable time limit clear and concrete. In other words, not only Arora et al.’s experimental methodology sets the bars unfairly for different algorithms (as described in Section 2.2.1), even the method for computing the bars (aka, “near-optimal” quality of each algorithm) is ill-defined.

To appreciate the gravity of this issue, consider a thought experiment where we have two exact replicas \mathcal{A}_1 and \mathcal{A}_2 of the same algorithm \mathcal{A} . Assume that Arora et al.’s experimental methodology is not aware that \mathcal{A}_1 and \mathcal{A}_2 are essentially the same algorithm. While tuning their parameters p for the two algorithms, if the same upper bound on running time T is not employed, then we will end up with two different values of p , namely $p_{\mathcal{A}_1}$ and $p_{\mathcal{A}_2}$ for the two replicas, respectively. W.l.o.g., suppose $p_{\mathcal{A}_1}$ is stricter than $p_{\mathcal{A}_2}$ (e.g., smaller accuracy threshold ϵ). As a result, \mathcal{A}_1 and \mathcal{A}_2 are necessarily held to different bars on spread and hence we may conclude that one of them is more efficient than the other. However, since \mathcal{A}_1 and \mathcal{A}_2 are replicas of each other, such a conclusion is absurd.

2.3 Irreproducible Results in Arora et al.’s Parameter Setting

Furthermore, we are unable to reproduce a vital experiment in [1] that was used to determine the parameter of each algorithm. We now elaborate on this.

Recall that Arora et al. set the parameter p of each algorithm A such that it yields a seed set whose expected spread is at least $\mu^* - sd^*$, where μ^* is the expected spread of the best seed set S^* that A can produce over a range of values of p , and sd^* is the standard deviation of S^* ’s spread in 10000 rounds of Monte-Carlo simulations. In other words, Arora et al.’s parameter setting for A highly depends on the measurement of μ^* and sd^* . In Figure 12 of [1], Arora et al. report their measurements of μ^* and sd^* for IMM, using seed set size $k = 200$ and varying the number of Monte-Carlo rounds from 1000 to 20000. Values of sd^* for other algorithms are not reported in their paper².

We attempted to reproduce Arora et al.’s results on two of the datasets that they use, i.e., Nethept and HepPH. On each dataset, we used their setting and ran IMM with $\epsilon = 0.05$ and $k = 200$ to obtain S^* , and then measured μ^* and sd^* using r rounds of Monte-Carlo simulations, with r varying from 1000 to 20000. We compute $\mu^* = \frac{1}{r} \sum_{i=1}^r I_i(S^*)$, where $I_i(S^*)$ denotes the spread of S^* in the i -th Monte-Carlo round. In addition, we compute sd^* using the standard formula for sample standard deviation:

$$sd^* = \sqrt{\frac{1}{r-1} \sum_{i=1}^r (I_i(S^*) - \mu^*)^2} \quad (2)$$

Figure 4 illustrates the values of μ^* and sd^* that we obtain from each dataset under the IC, WC, and LT models. On Nethept, the values of sd^* under the IC, WC, and LT models are approximately 50, 90, and 150, respectively, while on HepPH, the values of sd^* under the IC, WC, and LT models are roughly 60, 120, and 300, respectively. In contrast, the values of sd^* reported in Figure 12 of [1] (for 10000 Monte-Carlo rounds) are around *10 times* smaller than the corresponding values in Figure 4. For instance, in case of the IC model on Nethept, Arora et al. report that $sd^* < 5$. Such a small sd^* is anomalous given the large amount of randomness in Monte-Carlo simulations.

We note that such significant errors (10x gap) in the measurement of sd^* *have serious implications on all experimental results in [1]*: If sd^* is measured incorrectly, then the

²We requested the authors to share their code that computes standard deviations. Unfortunately, even after several attempts, we are still waiting for their code.

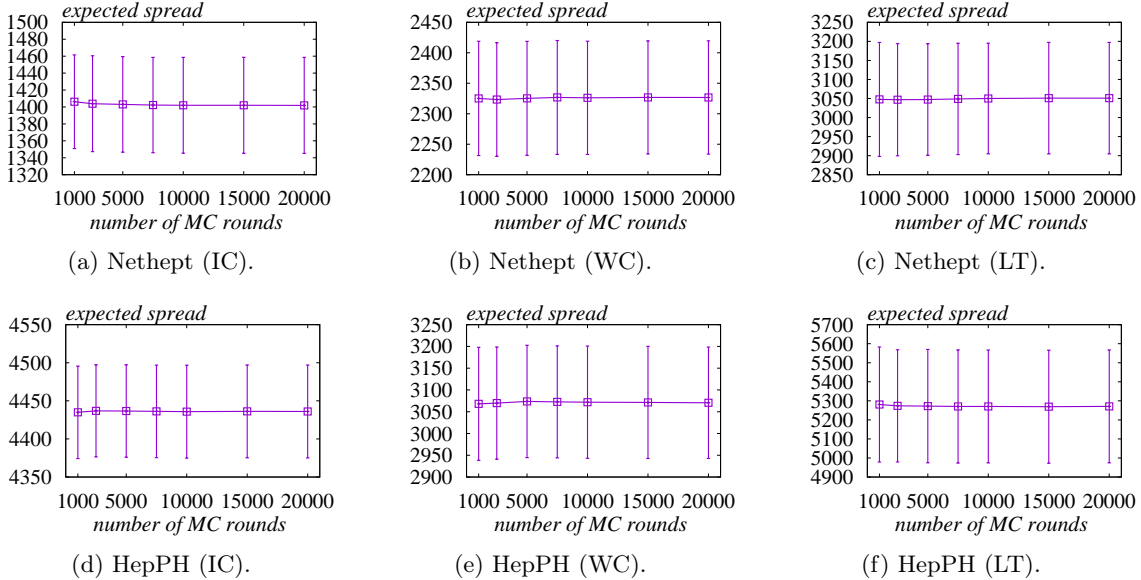


Figure 4: μ^* and sd^* on Nethept and HepPH.

parameter settings for all experiments in [1] are erroneous, since the latter are decided based on the former. To correct such serious errors, all experiments in [1] would have to be re-run. We elaborate further on this in the next para.

Reasons for Discrepancy. We corresponded with Arora et al. via email regarding the above issue on sd^* , and were told that the discrepancy is due to the fact that they performed “binning” and “smoothing” operations on the distribution before calculating sd^* . They claimed that such operations were done to “remove outliers”. It is noteworthy that there is no mention of binning and smoothing in the paper [1]. We requested them for a detailed explanation of how they performed binning and smoothing and/or the code they used for performing these operations. Despite multiple reminders, we are still waiting for their code and explanations.

Regardless of how this “binning” was conducted, we note that there are at least three serious issues at hand here. First, Eq (2) is the standard definition of sample standard deviations, and thus conducting any “binning” operations prior to the application of Eq (2) and still calling it “standard deviation” (i.e., without any qualification) is misleading at best. Second, the notion of “outliers” is irrelevant when measuring standard deviations because, for any random variable v sampled from a distribution Ω , the standard deviation of v is defined over all possible samples from Ω – that is, by definition, no sample from Ω should be ruled as outliers. Third, in the paper itself [1], there is no mention of the “binning” operations at all, which is rather strange given that they have such profound effects in the measurement of sd^* .

For instance, if no “binning” or “smoothing” is done and sd^* of IMM is taken from our Figure 4(d), then IMM would be allowed to run with a spread bar of 4380. From Figure 2, this implies that ϵ can be set at 0.35, with running time 29.8 sec, a 37.2x speed-up compared to $\epsilon = 0.05$!

3. REFUTATIONS ON ALGORITHM-SPECIFIC MISCLAIMS

Not only the paper has an ill-designed experimental methodology as highlighted above, it is also riddled with various misclaims. This section investigates some of the algorithm-specific issues and misclaims.

3.1 TIM⁺ vs. IMM

MISCLAIM 1. “Both TIM⁺ and IMM do not scale beyond HepPh in terms of memory-consumption under the IC model.” – [1], Section 5.2, Section 5.3.1, Section 5.4 (repeated).

Refutation: Arora et al. make this claim solely based on the running time of TIM⁺ and IMM when they set $\epsilon = 0.05$. This setting, as we mentioned in Section 2.3, requires TIM⁺ and IMM to generate extremely accurate results, which significantly increases the computation overheads of both algorithms. In contrast, other algorithms compared (e.g., EaSyIM) are allowed to produce less accurate results (due to the experimental methodology used in [1] – see Section 2.1), which leads to much smaller computation costs. If we lower the accuracy of IMM to the same level as other algorithms (e.g., EaSyIM), then it would become much more scalable. Therefore, the above claim is invalid.

MISCLAIM 2. “TIM⁺ is faster than IMM under LT model.” – [1], Section 5.3.1, Section 6, M3 (repeated).

Refutation: Arora et al. compare the empirical running time of TIM⁺ and IMM under the LT model by setting $\epsilon = 0.1$ for TIM⁺ and $\epsilon = 0.05$ for IMM, and they conclude that TIM⁺ is empirically more efficient than IMM under the LT model. Unfortunately, the comparison is meaningless and the conclusion made is invalid, as we explain in the following.

Both TIM⁺ and IMM rely on drawing subgraph samples (i.e., RR-sets [2]) for influence maximization. For both algorithms, the number of samples used is decided based on the given *theoretical worst-case accuracy threshold* ϵ , and the running time of the algorithms almost solely depends on the

sample number. IMM offers a tighter asymptotic bound of sampling accuracy than TIM⁺ does, and hence, for a given ϵ , it can use a smaller sample set than TIM⁺ to achieve a $(1 - 1/e - \epsilon)$ -approximation. In other words, IMM is more efficient than TIM⁺ when they are held to the same *worst-case accuracy threshold* ϵ , as convincingly demonstrated in [26].

In contrast, Arora et al. compare the efficiency of TIM⁺ and IMM under the LT model with $\epsilon = 0.1$ for TIM⁺ and $\epsilon = 0.05$ for IMM³, which is meaningless given the differences in the ϵ value, as different ϵ values represent different approximation guarantees of the solutions. A slightly more sensible comparison is to evaluate the computation cost of the two algorithms when they achieve the same *empirical accuracy*. However, in that case, the comparison is *moot* and should lead to only one conclusion: they have roughly the same running time when their empirical accuracies are the same. The reason is that the *empirical accuracy* of the two algorithms depends only the sizes of the sample sets that they use. If the experiment is set up in such a way that the two algorithms will yield the same empirical accuracy, then the number of samples that they use should be roughly the same, in which case their running time would also be similar. Therefore, it is incorrect to claim that TIM⁺ is more efficient than IMM under the LT model, regardless of whether the accuracy metric concerned is the empirical expected spread or the worst-case accuracy threshold ϵ .

Profound nature of Arora et al.’s mistakes. In fact, comparing the empirical accuracies of TIM⁺ and IMM is analogous to comparing the empirical accuracies of Chernoff’s bound and Chebyshev’s inequality when estimating the mean of a random variable. Such a comparison is not useful at all, since the empirical accuracy of the estimation would be the same as long as the sample set size is the same. Recall that Chernoff’s bound is tighter than Chebyshev’s inequality, if one aims to achieve a given theoretical *guarantee*.

Ironically, Arora et al.’s experimental design may even lead to an obviously incorrect conclusion that Chebyshev’s inequality is superior to Chernoff’s bound, as we explain in the following. Let x_1, x_2, \dots, x_n be i.i.d. samples drawn from some distribution over $[0, 1]$ with mean μ and variance σ^2 . Chebyshev’s inequality says that for any $c > 0$,

$$\Pr \left[\left| \frac{1}{n} \sum_{i=1}^n x_i - \mu \right| \geq c \right] \leq \frac{\sigma^2}{nc^2},$$

which is equivalent to

$$\Pr \left[\left| \frac{1}{n} \sum_{i=1}^n x_i - \mu \right| \geq \epsilon \mu \right] \leq \frac{\sigma^2}{n\epsilon^2\mu^2}.$$

Meanwhile, Chernoff’s bound says

$$\Pr \left[\left| \frac{1}{n} \sum_{i=1}^n x_i - \mu \right| \geq \epsilon \mu \right] \leq 2 \exp \left(-\frac{n\mu\epsilon^2}{3} \right).$$

In other words, from Chebyshev’s inequality, if we are to use $\frac{1}{n} \sum_{i=1}^n x_i$ as an estimation of μ and we aim to achieve ϵ

³The root cause of selecting different ϵ values goes back to their ill-designed experimental methodology, which is highlighted in Section 2

relative error with at least $1 - \delta$ probability, then the number n of samples required should satisfy

$$n \geq \frac{\sigma^2}{\delta \cdot \epsilon^2 \mu^2}.$$

Meanwhile, if we apply Chernoff’s bound instead, we have

$$n \geq \frac{3 \log(2/\delta)}{\epsilon^2 \mu}.$$

It can be verified that the number n from Chernoff’s bound is smaller whenever $3\delta\mu \log(2/\delta) \leq \sigma^2$. For example, Table 1 shows the sample numbers required by Chebyshev’s inequality and Chernoff’s bound when $\mu = \sigma = 1/2$, $\delta = 10^{-3}$, and for various values of ϵ .

Suppose that we are to compare the “empirical efficiency” of Chebyshev’s inequality and Chernoff’s bound, following Arora et al.’s methodology. In that case, we first fix δ (e.g., $\delta = 10^{-3}$) and, for Chebyshev’s inequality (resp. Chernoff’s bound), we set a threshold τ_1 (resp. τ_2) on the maximum absolute error allowed in the estimation of μ . (This is in the same spirit of requiring an influence maximization algorithm to achieve an expected spread of at least $\mu^* - sd^*$.) For simplicity, assume that $\tau_1 = \tau_2 = \tau$.

Next, we vary ϵ in $\{0.05, 0.1, 0.15, 0.20, \dots\}$. (Arora et al. use this set of ϵ values in their evaluation of TIM⁺ and IMM.) For each ϵ , we compute the number n of samples required by Chebyshev’s inequality (resp. Chernoff’s bound), and measure the empirical error in the estimation of μ when we use n samples from Ω . Suppose that ϵ_1 (resp. ϵ_2) is the largest value of ϵ for which Chebyshev’s inequality (resp. Chernoff’s bound) has an empirical error no more than τ . Then, we will compare the sample number n_1 required by Chebyshev’s inequality with $\epsilon = \epsilon_1$ against the number n_2 required by Chernoff’s bound with $\epsilon = \epsilon_2$, and we declare that Chebyshev’s inequality is empirically more efficient if $n_1 < n_2$.

For the sake of argument, assume that when we use at least 10000 samples from Ω , the estimation of μ has at most τ error with at least $(1 - \delta)$ probability. Then, in the context of Table 1, we can see that $\epsilon_1 = 0.3$ and $\epsilon_2 = 0.05$. Accordingly, $n_1 = 11112$ and $n_2 = 18243$, which leads to the conclusion that Chebyshev’s inequality is empirically “more efficient” than Chernoff’s bound!

3.2 SimPath vs. LDAG

We now address the misclaims made by Arora et al. [1] against the SimPath algorithm [17]. Importantly, while attempting to reproduce their experiments, we obtained results directly contradict [1], and are in support of the original SimPath paper [17]. We will explain the discrepancies. We have verified all results presented in this subsection on both NTU and UBC servers⁴.

3.2.1 Misclaims in Arora et al. & Refutations

MISCLAIM 3. “SIMPATH fails to finish even after 2400 hours on DBLP and YouTube.” – [1], Section 5.3.1 and Section 6, M5 (repeated).

Refutation: Unfortunately, Arora et al. failed to supply compatible datasets to the SimPath source code [15] released

⁴Unless otherwise stated, whenever running time numbers are mentioned in text, they are taken from the UBC results.

ϵ	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	...
Chebyshev	400000	100000	44445	25000	16000	11112	8164	6250	...
Chernoff	18243	4561	2027	1141	730	507	373	286	...

Table 1: Sample size required by Chebyshev’s inequality and Chernoff’s bound for various ϵ ($\mu = \sigma = 1/2, \delta = 10^{-3}$).

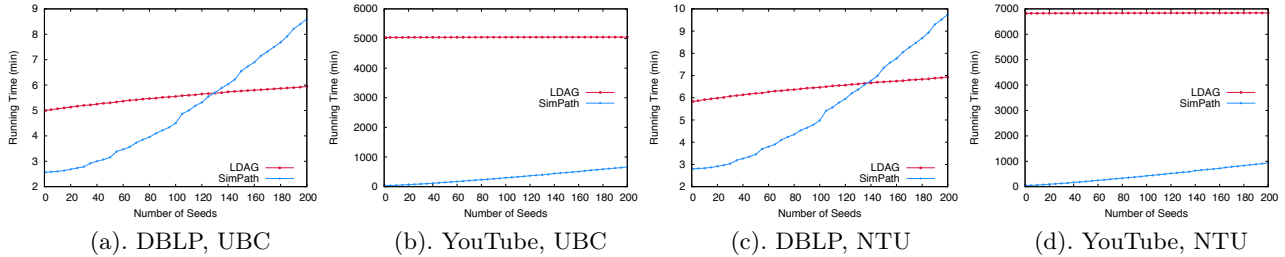


Figure 5: Running time of SimPath and LDAG, on DBLP and YouTube datasets. Note that the UBC server has faster CPUs and larger memory, and hence generally both algorithms run faster on the UBC server.

by Goyal et al. [17], and ran into *infinite loops* for at least 2400 hours. We have reached out to Arora et al., and they have acknowledged this issue. In particular, the code reserves node-id “0” for backtracking purposes, and therefore assumes that all input node-ids are positive integers. However, the datasets used by Arora et al. contain 0 as node-ids. They neither preprocessed the datasets correctly, nor did they attempt to debug it when their experiments were stuck in an infinite loop for 100 days.

With corrected input data, we verified that SimPath finishes within just a tiny fraction of 2400 hours: It took just 8.6 minutes on DBLP and 667 minutes on YouTube to select 200 seeds, representing 0.00597% and 0.463% of 2400 hours, respectively (Figure 5)!

MISCLAIM 4. “We discover that SIMPATH provides faster performance than LDAG only on the “parallel edges” LT model, which is used in the SIMPATH paper [15]. In our experiments, we use the “uniform” LT model.” – [1], Section 5.3.1 and Section 6, M5 (repeated).

Refutation: Since the two datasets, DBLP and YouTube, on which Arora et al. ran into infinite loops happen to be prepared according to the so-called “LT-uniform” model, Misclaim 4 is in fact an corollary of Misclaim 3, and thus incorrect.

MISCLAIM 5. “In the SIMPATH paper, these two techniques are not evaluated beyond 100 seeds and thus this observation remained hidden.” – [1], Section 6, M5.

Refutation: Almost all major IM work published prior to or shortly after SimPath [4–7, 18–20] conducted experiments with $k = 50$ seeds or less, including the LDAG paper itself [7]. Hence, Misclaim 5 is ignorant of the historical context of the research in this domain. Moreover, the very trend that the running time of LDAG grows slower than SimPath as k increases can be observed directly from the original SimPath paper (cf. Figure 4 in [17]). The usage of “hidden” by Arora et al. is misinformed and misleading.

MISCLAIM 6. “Overall, these results indicate that LDAG not only scales better than SIMPATH but is also more robust to the underlying diffusion model.” – [1], Section 6, M5.

Refutation: Our results on the YouTube dataset directly refute this misclaim: Figure 5 shows that when $k = 200$, SimPath took 667 minutes while LDAG took 5047 minutes, which amounts to a gap of 7.5x favoring SimPath. As refuted in Misclaim 4, the distinction between the so-called “LT-uniform” and “LT-parallel” models actually stems from an infinite loop, in turn the result of not preprocessing the data correctly. Therefore, LDAG being “more robust to the underlying diffusion model” is incorrect.

MISCLAIM 7. “Note that LDAG [6], IRIE [16] and SIMPATH [15] do not have any external parameters, and thus this analysis does not cover them.” – [1], Section 5.1.1.

Refutation: Arora et al. define the external parameter as follows⁵: It controls the quality and running time trade-off for the algorithm, and can be explicitly configured by the user. Unfortunately, the authors have overlooked the fact that both LDAG and SimPath do have such a parameter, and both papers clearly stated so.

- LDAG: Section IV.B, [7]: “... controlling the size of the LDAG using parameter θ , which represents a tradeoff between efficiency (smaller DAGs and thus faster computations) and accuracy (larger DAGs and more accurate influence result).”
- SimPath: Section I.A, [17]: “We propose a parameter η to control the size of the neighborhood that represents a direct trade-off between accuracy of spread estimation and running time.” In fact, Table III in [17] shows the trade-off between spread achieved and running time by varying parameter η , on two different datasets.

In the released code [15], both θ in LDAG and η in SimPath can be set or changed in a plain-text configuration file; no code change is required.

⁵The definition of “external parameters” can be seen from the following exact quotes in [1]: Section 3.1.1 states “Majority of IM algorithms M possess an external parameter which controls its accuracy... The more stringent the choice of this parameter the better the accuracy. Consequently, the more the running time.” In addition, Section 5.1.1 states “The external parameters are exposed through the API and can be tuned to optimize performance.”

3.2.2 Reasons for Discrepancies

As mentioned, Misclaims 3, 4, and 6 are caused by the fact that Arora et al. [1] did not use compatible datasets with our SimPath source code [15], and got stuck in an infinite loop for 100 days. To explain why an infinite loop may occur, first recall that the SimPath algorithm has a backtracking procedure (Algorithm 2 in [17]) to estimate expected influence spread. The code makes use of integer “0” as a dummy node id for resetting the FIFO queue in backtracking. If there exists a node in the graph with real node id 0, then whenever backtracking is called on this node, an infinite loop will occur because the code cannot distinguish between the real node id 0 and the dummy id 0.

It is worth noting that depending on the graph structure, not all nodes may reach the backtracking stage. Therefore, the mere presence of node id 0 in the input graph may not always lead to infinite loops. Arora et al. apparently ran into infinite loops for two datasets: DBLP and YouTube. Unfortunately, they did not attempt to understand or debug the code and instead chose to continue running the experiments for 2400 hours (100 days)⁶.

3.2.3 Experiments and Analysis

We ran experiments on the DBLP and YouTube datasets (“LT-uniform”) to attempt to reproduce Misclaims 3, 4 and 6. As mentioned earlier, we obtained results sharply contradicting the above misclaims.

The DBLP dataset contains 317K nodes and 1.05M undirected edges. The YouTube dataset contains 1.13M nodes and 2.99M undirected edges. See Table 1 in [1] for detailed statistics. In both datasets, the graphs are undirected. As with [1], we direct all edges in both directions and for each resultant directed arc (u, v) , the influence weight $p_{u,v} = 1.0/\text{indegree}(v)$ (as per the definition of “LT-uniform” in [1]). To avoid infinite loops, we preprocessed the data such that only positive integers are used as node ids.

We ran the experiments at both UBC and NTU, independently. The executors used the same source code, configurations, and input graph files. The UBC server runs OpenSuSE, has 16-core Intel Xeon X5570 CPUs at 2.93GHz each, and 94.4GB RAM. The NTU server runs Debian, has 6-core Intel Xeon CPUs E5645 at 2.40GHz each, and 32GB RAM. Every experiment (one particular algorithm on one particular dataset) was run as the only active process on the server, except for those required by the operating system.

Results and Analysis. As with [1], we set $k = 200$. The running time obtained on the UBC and NTU servers are shown in Figure 5. Raw numbers can be viewed in our public GitHub repository⁷. The trends (growth of running time as k increases) are consistent on both datasets. The UBC server has faster CPUs and larger main memory, and hence the absolute values of the running time are smaller in Figure 5(a)(b) than Figure 5(c)(d). To re-iterate, our results here directly and clearly refute Arora et al.’s misclaims on SimPath’s running time.

⁶We would like to gently remind the reader that any unsupported experimental code can encounter unforeseen issues, and users are encouraged to understand the code instead of applying it blindly. In case of unexpected results, debugging helps. Typically no warranty or author liability is attached to open source code, as in the case for our code as well.

⁷<https://github.com/jjboo/simpath-results>

- Refutation on Misclaim 3: SimPath took just 8.6 minutes on DBLP and 667 minutes on YouTube for selecting 200 seeds, representing 0.00597% and 0.463% of 2400 hours, respectively.
- Refutation on Misclaim 4: Both DBLP and YouTube datasets here are prepared under the “LT-uniform” definition. Figure 5 shows that SimPath is clearly the winner on the larger YouTube data, taking 667 minutes to select 200 seeds, while LDAG takes 5047 minutes. On DBLP data, SimPath is faster initially and LDAG catches up after approximately 130 seeds.
- Refutation on Misclaim 6: The 7.5x gap between SimPath’s (667 minutes) and LDAG’s (5047 minutes) running time on YouTube directly refutes Misclaim 6. While LDAG is indeed a significant algorithmic contribution to IM under the LT model, it takes more than 3 days to finish selecting 200 seeds on a graph with 1.1M nodes whereas SimPath finished in 667 minutes.

Further Remarks. This exercise in fact illustrates different natures of SimPath and LDAG, and their respective advantages. LDAG does most of its work upfront, constructing a local directed acyclic graph for each node before going on to mine seeds. Once the local DAGs are built, the seed selection process is relatively fast. The SimPath algorithm, on the other hand, estimates the expected influence spread directly on the original input graph and intelligently enumerates and prunes simple paths to achieve this goal. As the seed set size increases, the number of paths that SimPath needs to examine increases, resulting in larger running time. Given the very different nature of LDAG and SimPath, it is not surprising at all that on certain datasets, LDAG may regain advantages as the seed set size becomes larger.

4. REFUTATIONS ON OTHER MISCLAIMS

In addition to the list of serious issues discussed extensively in Sections 2 and 3, [1] contains several other incorrect, misleading, and/or unscientific statements. Next, we shall give a *non-exhaustive* list and discuss them in detail.

4.1 Misclaims about EaSyIM [13]

EaSyIM is an algorithm proposed by Galhotra, Arora, and Roy in SIGMOD 2016 [13]. Arora et al. [1] make the following claims on the superiority of EaSyIM:

MISCLAIM 8. “EaSyIM [10] is most memory-efficient ... EaSyIM only stores a number per node. Consequently, it is the most memory efficient technique for IM.” – [1], Section 5.4.

MISCLAIM 9. “Fig. 11b presents the decision tree for choosing the best IM technique given the task and resources in hand... When main memory is scarce, EaSyIM, CELF, CELF++ and IRIE provide alternative solutions. Among these, EaSyIM easily out-performs the other three techniques in memory footprint, while also generating reasonable quality and efficiency. Overall, the choice is between four techniques: IMM, TIM⁺, EaSyIM, and PMC.” – [1], Section 7

Refutation. Arora et al. [1] helpfully summarize their empirical findings in the form of a decision tree which is intended to make recommendations for what IM algorithm

to use under what circumstances. Let us examine a couple of recommendations coming out of their decision tree and compare those recommendations with common sense.

Consider the IM problem under the LT model over a large dataset. Suppose available main memory is large. Then according to their decision tree, one should use TIM^+ and not IMM. We have already established in Section 3.1 that under all major diffusion models including LT, IMM strictly dominates TIM^+ in the sense that for any desired *theoretical worst case guarantee* ϵ , IMM can deliver a seed set S of size k whose expected spread is guaranteed to be at least $(1 - 1/e - \epsilon) \cdot OPT$, w.h.p., with far fewer RR-sets than TIM^+ can. While TIM^+ can provide the same guarantee, it comes at the price of many more RR-sets than needed by IMM. If following Arora et al.’s argument we go ahead and use TIM^+ instead of IMM, it is true that *sometimes* one may obtain the same *empirical* accuracy as IMM with fewer RR-sets than dictated by the worst-case guarantee. But this accuracy is not always guaranteed. Thus, this recommendation can lead to a poor choice of algorithm.

As a second example, in that decision tree, Arora et al. recommend EaSyIM [13] as the best choice for all three diffusion models (i.e., IC, WC, and LT) when “memory is scarce”, and claim that EaSyIM is “generating reasonable quality and efficiency” without rigorously defining what exactly is meant by “reasonable”. This recommendation is misleading and questionable for several reasons.

First, Table 3 in [1] itself presents contradicting results: For the WC and LT settings, EaSyIM “did not terminate even after 40 hours”⁸ on Orkut, Twitter, and Friendster. The only dataset in Table 3 that EaSyIM can handle is LiveJournal, which is only 2.5GB in size. This demonstrates that EaSyIM does not really offer reasonable efficiency or scalability, certainly not comparable to IMM or TIM^+ which finish on all datasets. Second, given that EaSyIM cannot handle the datasets larger than 2.5GB under WC and LT, it cannot really be recommended whenever “memory is scarce”. For example, consider that we have a 4GB dataset and 5GB memory. The memory size is relatively small in comparison to the dataset size, and yet, EaSyIM would not be able to process the data due to its excessive computation cost. Third, given that nowadays, desktop computers (resp. workstations) can easily have more than 8GB (resp. 32GB) of memory, it is unrealistic to assume that one needs to process a smaller-than-2.5GB dataset with a machine whose memory is small with respect to the data.

Further, Misclaim 8 declares *in absolute terms* that EaSyIM [13] is the most memory-efficient technique for IM. The rationale is that EaSyIM requires just a single number to be stored per node. So, by the same argument, one could prefer the Random algorithm, which randomly chooses k seeds, over *all* algorithms, including EasyIM, since Random does not require any information to be stored per node. At another extreme, basic Greedy with MC simulations can also be chosen over EasyIM, as it also stores only one number per node. Consider the extreme nature of these choices. Random is extremely fast but can lead to poor spread. Greedy with MC simulations can lead to very high spread depending on the number of simulations employed, but is extremely slow. Thus, an argument for choosing an algorithm solely based on its low memory consumption is ill-conceived as it

⁸Exact quote from the caption of Table 3 in [1].

ignores other equally important factors like running time and spread.

4.2 Misleading “Myths” and Other Issues

MISCLAIM 10. “M2. CELF (or CELF++) is the gold standard for quality” – [1], Section 6, M2.

Refutation. This is a myth Arora et al. [1] claimed to have found from the influence maximization literature. Their argument is that the solution quality of CELF (or CELF++) depends on the number of Monte Carlo (MC) simulations. First of all, CELF and CELF++ are both heuristic optimizations on top of the vanilla greedy approximation algorithm, that save on the number of marginal gain computations and have nothing to do with “quality”. It is well known that the quality solely depends on the number of MC simulations, which are used to compute the spread function, and which are orthogonal to the CELF and CELF++ optimizations. Hence, CELF or CELF++, per se, has nothing to do with quality. In particular, it is strange to suggest that a heuristic, for saving on marginal gain computations, in and of itself can be a standard of anything leave alone gold standard of quality!

MISCLAIM 11. “M6. WC is equivalent to IC. Several techniques have misused the term IC ... they claimed to be the state-of-the-art for IC. In reality, they all fare poorly on the generic IC model ...” – [1], Section 6, M6.

Refutation. It is correct that IC and WC indeed have different meanings. However, the comments of Arora et al., quoted above, are misinformed and misleading.

IC refers to the “Independent Cascade” model, while “WC” stands for “Weighted Cascade”, which is one method to compute influence probabilities under the IC model, i.e., $p_{u,v} = 1/\text{indegree}(v)$. One can use the WC method to compute influence probabilities, and find seeds in the IC model. That said, there does exist an early work [6] (2009) that freely used “IC” to refer to the two things collectively. Consider the following two methods of assigning influence probabilities: (1) the WC method; (2) assigning an identical influence probability of 0.1 to all edges in the graph. Both (1) and (2) are just two different methods of assigning probabilities. Neither of them is generic. In particular, method (2) grossly oversimplifies reality as there is no reason to believe that all edges will have the same influence probability! Surprisingly, Arora et al. refer to (2) “*general IC model!*” and go on to use this as a tool to criticize many prior papers.

Finally, regarding the experimental platform used in [1], Arora et al. state that in Section 5 (emphasis added): “*All experiments are performed using codes written in C++ on an Intel(R) Xeon(R) E5-2698 64-core machine with 2.3 GHz CPU and 256 GB RAM running Ubuntu 14.04.*” and “*IRIE [16] was compiled on a Microsoft Windows 7 machine possessing the same configuration.*”

Observations & Remarks. Essentially, Arora et al. [1] ran one algorithm (IRIE) on Windows 7 while all other algorithms on Ubuntu (Linux). Then the running time and memory usage of IRIE are squarely compared together with all other Linux-run algorithms (see Figure 7 and 8 of [1]). This is clearly an unscientific approach to benchmarking. At the very least, a benchmarking experiment should be

done using the *same* operating system platform. This is especially critical for C++ as the compilation and optimization of C++ code are vastly different for Linux (e.g., gcc) and Windows (e.g., Microsoft Visual Studio).

5. CONCLUSIONS

The benchmarking paper by Arora et al. [1] claims to unearth and debunk so-called “myths” that allegedly propagated in the Influence Maximization research literature over the years. However, our refutations show that not only their experimental methodology is ill-designed and flawed, many of their specific experimental results and claims are incorrect. In particular, they fail to incorporate the trade-off between running time and the seed set quality correctly in their experimental methodology. In addition, they fail to distinguish between theoretical guarantee on spread from empirical spread, resulting in misleading and often wrong recommendations of IM algorithms to use under given resource constraints.

In this paper, we have systematically refuted the experimental methodology of Arora et al. and 11 of the specific claims made in their paper.

Acknowledgements

We sincerely thank Yu Yang (SFU) and Jian Pei (SFU) for their feedback on an earlier version of this work, and for contributing the counterexample in Figure 3 that depicts a flaw in the experimental methodology of Arora et al. [1].

6. REFERENCES

- [1] A. Arora, S. Galhotra, and S. Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. In *SIGMOD*, page to appear, 2017.
- [2] C. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, pages 946–957, 2014.
- [3] W. Chen, L. V. S. Lakshmanan, and C. Castillo. *Information and Influence Propagation in Social Networks*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.
- [4] W. Chen, W. Lu, and N. Zhang. Time-critical influence maximization in social networks with time-delayed diffusion process. In *AAAI*, 2012.
- [5] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010.
- [6] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.
- [7] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*, pages 88–97, 2010.
- [8] S. Cheng, H. Shen, J. Huang, W. Chen, and X. Cheng. Imrank: influence maximization via finding self-consistent ranking. In *SIGIR*, pages 475–484, 2014.
- [9] S. Cheng, H. Shen, J. Huang, G. Zhang, and X. Cheng. Staticgreedy: solving the scalability-accuracy dilemma in influence maximization. In *CIKM*, pages 509–518, 2013.
- [10] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM*, pages 629–638, 2014.
- [11] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.
- [12] N. Du, L. Song, M. Gomez-Rodriguez, and H. Zha. Scalable influence estimation in continuous-time diffusion networks. In *NIPS*, pages 3147–3155, 2013.
- [13] S. Galhotra, A. Arora, and S. Roy. Holistic influence maximization: Combining scalability and efficiency with opinion-aware models. In *SIGMOD*, pages 743–758, 2016.
- [14] M. Gomez-Rodriguez, L. Song, N. Du, H. Zha, and B. Schölkopf. Influence estimation and maximization in continuous-time diffusion networks. *ACM Trans. Inf. Syst.*, 34(2):9:1–9:33, 2016.
- [15] A. Goyal, W. Lu, and L. V. Lakshmanan. Ubc influence maximization software. <http://www.cs.ubc.ca/~welu/downloads.html>. Accessed: 2017-05-07.
- [16] A. Goyal, W. Lu, and L. V. S. Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *WWW*, 2011.
- [17] A. Goyal, W. Lu, and L. V. S. Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *ICDM*, pages 211–220, 2011.
- [18] K. Jung, W. Heo, and W. Chen. Irie: Scalable and robust influence maximization in social networks. In *ICDM*, 2012.
- [19] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.
- [20] M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In *PKDD 2006*, Lecture Notes in Computer Science, pages 259–271. Springer Berlin / Heidelberg.
- [21] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.
- [22] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.
- [23] H. T. Nguyen, M. T. Thai, and T. N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *SIGMOD*, pages 695–710, 2016.
- [24] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi. Dynamic influence analysis in evolving networks. *PVLDB*, 9(12):1077–1088, 2016.
- [25] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.
- [26] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: a martingale approach. In *SIGMOD*, pages 1539–1554, 2015.
- [27] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: near-optimal time complexity meets practical

efficiency. In *SIGMOD*, pages 75–86, 2014.

- [28] Y. Yang, X. Mao, J. Pei, and X. He. Continuous influence maximization: What discounts should we offer to social network users? In *SIGMOD*, pages 727–741, 2016.
- [29] Y. Yang and J. Pei. A note on using $\mu - sd$ as a lower bound of expected spread in influence maximization. Personal communication, May 2017.

APPENDIX

A. BACKGROUND ON INFLUENCE MAXIMIZATION

Two classical stochastic propagation models were studied by Kempe et al. in [19]: *Independent Cascade (IC)* and *Linear Thresholds (LT)*, both of which originally stem from mathematical sociology. For the exact definitions of IC and LT models, we refer the reader to [3, 19]. Kempe et al. [19] showed that the problem of Influence Maximization (IM) is NP-hard under both models. Further, Chen et al. [5, 7] showed that under both models, it is #P-hard to compute the exact value of the expected influence spread $\sigma(S)$, for any seed set S . As a result, IM is a computationally challenging problem. Kempe et al. [19] established that the spread function $\sigma(\cdot)$ is *monotone*⁹ and *submodular*¹⁰ under both IC and LT models. Hence, by applying a seminal result in Nemhauser et al. [22], one can use a simple greedy hill-climbing style algorithm to achieve an approximation factor of $(1 - 1/e - \epsilon)$, for any $\epsilon > 0$. The existence of ϵ is because it is #P-hard to compute the spread function $\sigma(\cdot)$ exactly, and hence Monte Carlo (MC) simulations need to be used jointly with the greedy algorithm. The greedy approximation algorithm starts with $S = \emptyset$, and runs for k iterations. In each iteration, it selects into S a node that yields the largest marginal gain, defined as $\sigma(S \cup \{u\}) - \sigma(S)$. We refer to this as the simple *Greedy algorithm with MC simulations*.

The computational costs associated with MC simulations are inevitably high, rendering the greedy algorithm quite inefficient and unable to scale to large graphs. Several notable improvements have been made since then [2, 3, 5, 7–9, 12, 14, 17, 18, 21, 24, 26–28], some of which are covered in this paper. We next recall them briefly.

CELF [21]. The vanilla greedy algorithm coupled with MC simulations, is rather simplistic. In each iteration, for all $w \in V \setminus S$ (all nodes in the graph except for those already selected as seeds), a re-computation of its marginal gain $\sigma(S \cup \{w\}) - \sigma(S)$ is done. CELF, for Cost-Effective Lazy-Forward, employs a clever optimization technique: It sorts nodes in non-increasing order of their latest marginal gain value (in a max-heap), and only recomputes the marginal gain of a node when this particular node surfaces to the root of the max-heap.

CELF++ [16]. A poster paper by Goyal et al. [16] built on top of CELF and tried to further improve the greedy algorithm by leveraging submodularity “more aggressively”. Whenever an evaluation of marginal gain is carried out, CELF++ also does a speculative, look-ahead marginal gain

⁹A set function f is *monotone* if $f(S) \leq f(T)$ whenever $S \subseteq T$.

¹⁰A set function f is *submodular* if $f(S \cup \{x\}) - f(S) \geq f(T \cup \{x\}) - f(T)$, for all S and T where $S \subseteq T$ and all $x \notin T$.

computation. More specifically, for each node w , CELF++ records the current best node in the max-heap (let us call it w^*) and computes $\sigma(S \cup \{w\}) - \sigma(S)$ and $\sigma(S \cup \{w, w^*\}) - \sigma(S \cup \{w^*\})$ together. Then, by definition of submodularity, as long as w^* indeed gets chosen as a seed, $\sigma(S \cup \{w, w^*\}) - \sigma(S \cup \{w^*\})$ will be readily available. Notice that both CELF and CELF++ are heuristics that help save on the number of marginal gain computations and are completely orthogonal to the MC simulations. It is obvious that the quality of expected spread estimation is controlled by the # MC simulations used and has nothing to do with CELF or CELF++.

LDAG [7]. Chen et al. [5] proposed the LDAG (stands for *Local Directed Acyclic Graph*) heuristic for solving IM under the Linear Threshold (LT) model. The main intuitions is that (1). even though computing exact influence spread is #P-hard in the LT model, it can be done in linear time on DAGs; (2). as a heuristic, the influence propagates to and from a node can be estimated using a local neighborhood surrounding the node, instead of the whole graph. Hence, a two-phase heuristic algorithm, LDAG, was proposed. First, for each node $v \in V$, one local DAG structure is constructed. Second, seeds are selected in greedy order with marginal gains estimated using the DAGs. LDAG achieved impressive results – it was shown to be orders of magnitude faster than the greedy algorithm (with CELF), without sacrificing much solution quality.

SimPath [17]. LDAG has a few limitations, e.g., memory consumption and the slight aggressive nature of keeping only one DAG per node and allowing influence to only flow within that lone DAG. Goyal et al. [17] proposed a new heuristic called SimPath, establishing that under the LT model, influence spread can be computed by enumerating simple paths starting from the seeds. SimPath employs several heuristical optimizations to prune and cut off path enumerations, maintaining a balance between running time and seed set quality. In their experiments, Goyal et al. [17] showed that SimPath outperforms LDAG in three metrics: spread achieved (quality), running time, and memory consumption.

TIM and TIM+ [27]. The latest break-through in IM (approximation) algorithm design started from the notion of Reverse-Reachable sets (or, RR-sets in short)¹¹ by Borgs et al. [2]. Utilizing RR-sets, Borgs et al. developed an approximate algorithm with near-optimal time complexity, but it incurs significant computation overheads in practice.

Building upon Borgs et al.’s work, Tang et al. [27] designed the Two-phase Influence Maximization (TIM) algorithm that improves Borgs et al.’s solution in terms of both time complexity and practical efficiency. Given any IM instance, the first phase computes a lower-bound on the optimal spread, which is in turn used to determine the number of RR-sets, θ , that it should sample. The second phase, it samples θ RR-sets and returns a cardinality- k node-set which covers the most sampled RR-sets, as the seed set. For both IC and LT models produces, TIM provides $(1 - 1/e - \epsilon)$ -approximate solutions with at least $1 - |V|^{-\ell}$ probability.

TIM+ is an improved version of TIM. It has an intermediate step to refine the estimation of the lower bound on

¹¹Consider a deterministic graph $G = (V, E)$ (i.e., edge presence is binary instead of being probabilistic), and any arbitrary node $v \in V$. An RR-set, rooted at v , is the set of all nodes that can reach v .

the optimal spread, and hence yields a smaller θ (which means fewer samples and better efficiency). Both TIM and TIM⁺ have expected time complexity $O((k + \ell)(|E| + |V|) \log |V| \epsilon^{-2})$ and are orders of magnitude faster than the greedy algorithm. They achieve even better efficiency than fast heuristics such as SimPath [17] and IRIE [18].

IMM [26]. IMM is an improvement of TIM⁺ that offers significantly higher efficiency in practice while retaining the latter’s asymptotic guarantees (i.e., IMM also returns $(1 - 1/e - \epsilon)$ -approximations in $O((k + \ell)(|E| + |V|) \log |V| \epsilon^{-2})$ expected time). Its main difference from TIM⁺ is that it adopts a more advanced martingale-based approach to derive a tighter lower bound on the optimal spread, which enables it to achieve the desired approximation guarantee with a smaller number of RR sets than TIM⁺. Tang et al. show that IMM is up to 100 times faster than TIM⁺ when they achieve the same asymptotic assurance.

B. A NOTE ON CELF VS. CELF++

Arora et al. [1] in their experiments observed that there are no significant differences in running times of CELF and CELF++, contradicting the observations in the original CELF++ poster paper [16]. To verify their claims, we reimplemented both CELF and CELF++ algorithms. In particular, we observed that there exists non-trivial variance in the running times of both CELF and CELF++. After careful analysis, we observed that several factors contribute to this variance: (i) Many nodes tend to have very close spread or marginal gains (on the two settings we tried). Different runs of the same algorithm can make different choices while selecting seeds, making the rest of the seed selection computations very different. (ii) Different initial starting values for random number generation can lead to different choices, and thereby different running times. (iii) The running time varies significantly if multiple experiments are run at the same time, even on a multi-core server, as in many CPU designs, the L2 and L3 caches may be shared across all cores.

Therefore, in order to obtain accurate running time and a statistically significant result, it is required that only one experiment is run at a time, even if the machine has multiple cores and each experiment is run multiple times to get a distribution of running times. A statistical significance test can then be applied to determine if the difference in running time is statistically significant.

In the experiments we report here, to avoid these issues, we ran exactly one experiment at a time on a machine¹². In addition, to maintain fairness in comparisons, we randomly selected 10 starting values for random number generation that were used both in CELF and CELF++ in their 10 respective independent runs¹³. The results are shown in Figure 6. From 10 runs, under NetHEPT WC settings, the average running times of CELF and CELF++ are almost identical - 54.2 min and 55.7 min respectively. The p -value from t -test is 0.58 confirming that the difference is not statistically significant. Recall that a lower p -value implies higher significance in difference. Commonly, a p -value of 0.05 or lower is

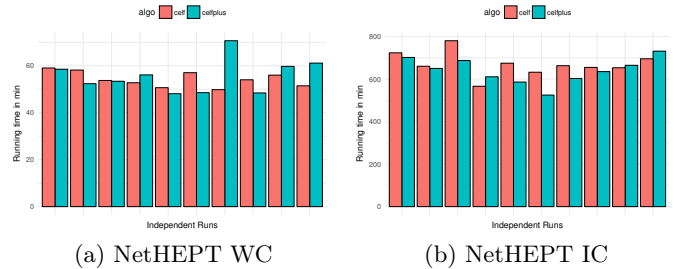


Figure 6: Running time CELF and CELF++ on NetHEPT, 10 independent & isolated runs each

considered suitable for establishing the difference to be statistically significant. In case of NetHEPT IC, with influence probabilities 0.1 on all edges, average CELF running time is 670.5 minutes while it is 639.5 minutes for CELF++. Here, CELF++ is faster than CELF by 4.6%, but the p -value is 0.25, making it not statistically significant either.

We hereby acknowledge that results reported in CELF++ poster paper [16] ran into noise and are not statistically significant. We would like to thank Arora et al. [1] for bringing up the issue to our attention.

C. IMPORTANCE OF UNDERSTANDING SOURCE CODE

Two exact quotes from Arora et al.:

- (a) “In this study, we need to either gather code from the authors or re-implement them.” – [1], Section 1
- (b) “Furthermore, to integrate them into the benchmarking framework and interpret the results, it is critical to have an in-depth understanding of the code.” – [1], Section 1

Our Observations. Even though statement (a) is not logically incorrect per se (due to the use of “or”), it is not clear from [1] which algorithm that Arora et al. had to “re-implement”. The implementation of EaSyIM [13] may come from Arora or Galhotra, but for the purpose of this paper, no re-implementation is needed. From Arora et al.’s shared repo¹⁴, all other algorithms’ implementations were made available to them by other research groups [7–9, 16–18, 24, 26, 27].

Regarding statement (b), unfortunately it appears that while Arora et al. highlighted the importance of having an in-depth understanding of the code, they themselves did not adhere to it. In particular, as described above, a couple of their SimPath experiments were stuck in infinite loop for 100 days, and yet, they did not attempt to understand the code or prepare the dataset properly during that whole time period.

¹²In fact, all experiments reported in this paper are run in isolation: only one experiment is run on a machine at a time.

¹³All code, including the algorithm to select starting values of random number generation is available at our GitHub repository: <https://github.com/jjboo/InfMax>. The MIT license applies.

¹⁴Shared by Arora et al: <https://drive.google.com/drive/u/1/folders/OB3hfiezv112RUEVVU1praGRQNWc>. Last accessed on May 12, 2017. A zipped copy of all files (version as of May 12, 2017) contained in the linked Google Drive folder is available upon request should this link become inactive.