Online Topic-aware Influence Maximization Queries

Cigdem Aslay[1] Nicola Barbieri[2] Francesco Bonchi[2] Ricardo Baeza-Yates[2]

[1]Web Research Group
University Pompeu Fabra
Barcelona, Spain
aslayci@acm.org

[2]Yahoo Labs
Barcelona, Spain
{barbieri,bonchi}@yahoo-inc.com
rbaeza@acm.org

ABSTRACT

Influence maximization is the key algorithmic problem behind viral marketing: it requires to identify a set of influential users in a social network, who, when convinced to adopt a product, shall influence other users in the network, leading to a large number of adoptions. Although real world users evidently have different degrees of interest and authoritativeness on different topics, the bulk of the literature on influence maximization is topic-blind, in the sense that it treats all items as they were the same.

In this paper we study *Topic-aware Influence Maximization* (TIM) queries: given a directed social graph, where the arcs are associated with a topic-dependent user-to-user social influence strength, and given a budget k, the problem requires to find a set of k users (named *seed set*) that we shall target in a viral marketing campaign for a given new item (described as a distribution over topics) in order to maximize its adoption. Our goal is to answer such queries in milliseconds, thus enabling online social influence analytics, what-if simulation, and marketing decision making.

The main challenge here is the enormous number of potential queries: any possible distribution over the topic space (i.e., any possible item) induces a different probabilistic graph, and thus a different instance of the standard influence maximization problem, for which efficiency and scalability are still unsolved problems.

Given these computational challenges, we propose to build an index over pre-computed solutions for a limited number of possible queries. Our proposal, INFLEX, employs a tree-based index for similarity search with Bregman divergences, to efficiently retrieve a good-enough set of neighbor points for the query item. Then it performs rank aggregation on their seed sets to produce the final answer to the query. Experimental results on real data show that INFLEX can provide in few milliseconds a solution very similar (Kendall-τ distance < 0.1) to the one produced by the best known *offline* computation (which usually takes several days).

1. INTRODUCTION

Viral marketing, a popular concept in business literature, has recently attracted a lot of attention also in computer science, thanks to the fascinating computational challenges that it entails. The idea of viral marketing is extremely appealing: by "targeting" the most influential users in a social network, we can exploit (fueled by word-of-mouth) the power of the network effect, thus delivering our marketing message to a large portion of the network through a self-replicating viral process, analogous to the spread of a virus.

In this area, the most studied computational problem, known as *influence maximization*, requires the identification of a set of k influential users (usually called the "seed set"), that should be targeted by the viral marketing campaign. Here, targeting might mean to give a free sample of a product, a special promotion, or a big discount. In order to enjoy the special promotion, the targeted user has to accept to automatically re-post it on her timeline over the social networking platform, so that her followers are exposed to the same marketing message.

The bulk of the literature on this problem just focuses on a generic item, thus implicitly assuming that the influence among users of the social network remains the same, independently of the characteristics of the item being propagated. Of course this is a very restrictive assumption: users' authoritativeness, expertise, trust, and influence are evidently topic-dependent. In this paper we drop such assumption and study how to provide a good seed set for a specific item in an online fashion. In order to formally define the problem studied, we first need to provide some background.

1.1 Background and related work

Kempe *et al.* [16] formalized the influence maximization problem based on the concept of propagation model: i.e., a stochastic model that governs how users influence each other and thus how propagations happen. Given a propagation model and a set of nodes $S \subseteq V$, the expected number of nodes "infected" in the viral cascade started with S, is called *(expected) spread* of S and denoted by $\sigma(S)$. The influence maximization problem asks for a set $S \subseteq V$, $|S| = k$, such that $\sigma(S)$ is maximum, where k is an input parameter.

The most studied propagation model is the so called *Independent Cascade* (IC) model. We are given a directed social graph $G = (V, A)$ with arcs $(u, v) \in A$ labeled by influence probabilities $p_{u,v} \in (0, 1]$, representing the strength of the influence of u over v. If $(u, v) \notin A$, we define $p_{u,v} = 0$. At a given time step, each node is either active (an adopter of product) or inactive. At time 0, a set S of seeds are

10.5441/002/edbt.2014.28

activated. Time unfolds deterministically in discrete steps. When a node u first becomes active, say at time t, it has one chance at influencing each inactive neighbor v with probability $p_{u,v}$, independently of the history thus far. If the attempt succeeds, v becomes active at time $t+1$.

Influence maximization is generally **NP**-hardḢowever Kempe *et al.* [16] show that the function $\sigma(S)$ is *monotone*[1] and *submodular.*[2] When equipped with such properties, the simple greedy algorithm that at each iteration extends the set of seeds with the node providing the largest marginal gain, produces a solution with provable approximation guarantee $(1-1/e)$ [21].

Though simple, the greedy algorithm is computationally prohibitive, since the step of selecting the node providing the largest marginal gain is #**P**-hard. In their paper, Kempe *et al.* run Monte Carlo simulations for sufficiently many times to obtain an accurate estimate of the expected spread. However, running many propagation simulations is extremely costly on very large real-world social networks. Therefore, following [16], considerable effort has been devoted to develop methods for improving the efficiency and scalability of influence maximization [17, 6, 13, 14]. Regardless of such research efforts, scalability still remains an open challenge: on the problem instances that we consider in this paper, even a state-of-the-art algorithm as CELF++ [14], takes from few days to more than a week in order to extract a seed set of 50 nodes. Most of this literature on efficient algorithms for influence maximization assumes the weighted social graph given, and do not address how the link influence probabilities $p_{u,v}$ can be obtained. This problem instead is addressed in [24, 26, 12].

Regardless of the fact that users authoritativeness, expertise, trust, and influence are evidently topic-dependent, only few papers have looked at social influence from the topics perspective. Tang *et al.* [26] study the problem of learning user-to-user topic-wise influence strength. The input to their problem is the social network and a prior topic distribution for each node, which is given as input and inferred separately. Liu *et al.* [19] propose a probabilistic model for the joint inference of the topic distribution and topic-wise influence strength: here the input is an heterogenous social network with nodes that are users and documents. The goal is to learn users' interest (topic distribution) and user-to-user influence. Lin *et al.* [18] study the joint modeling of influence and topics, by adopting textual models. None of these three papers define an influence propagation model, until a recent work by Barbieri *et al.* [2] extended the classic IC model to be topic-aware: the resulting model is named Topic-aware Independent Cascade (TIC) .

Barbieri *et al.* also devise methods to learn, from a log of past propagations, the model parameters, i.e., topic-aware influence strength for each link and topic-distribution for each item. Their experiments show that (1) topic-aware influence propagation models are more accurate in describing real-world influence driven propagations than the state-of-the-art topic-blind models, and (2) by considering the characteristics of the item, a larger number of adoptions can be obtained in the influence maximization problem. The TIC model is assumed at the basis of our work and it is introduced in detail next.

[1]$\sigma(S) \leq \sigma(T)$ whenever $S \subseteq T$.
[2]$\sigma(S \cup \{w\}) - \sigma(S) \geq \sigma(T \cup \{w\}) - \sigma(T)$ whenever $S \subseteq T$.

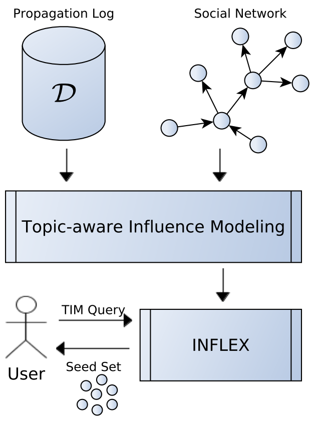


Figure 1: Topic-aware influence parameters are learnt from the log of past propagations and the social network following [2]. These are the prerequisites to build the INFLEX framework that we use to efficiently answer TIM queries.

1.2 Problem definition

Consider an advertisement platform allowing to implement "viral ads" over social networks, where users, by clicking on engaging ads, make them propagate to their friends. Advertisers come to the platform with a description of the ad (e.g., a set of keywords) to be promoted and they compete for the attention of the users which are considered influential w.r.t. the given description. In this kind of setting, not only it is important to consider the given description of the item for selecting the seed set appropriately, but such a decision must also be taken in an online fashion.

In this paper we study the online evaluation of *Topic-aware Influence Maximization* (TIM) queries. We are given a directed social graph $G = (V, A)$ and a space of Z topics. We assume the TIC propagation model introduced in [2], whose parameters are learned from a log of past propagation traces. In particular for each arc $(u, v) \in A$ and for each topic $z \in [1, Z]$ we have a probability $p^z_{u,v}$ representing the strength of influence that user u exerts over user v for topic z. A high level depiction of our setting is provided in Figure 1. An item i is described by a distribution $\vec{\gamma}_i$ over the topics: that is, for each topic $z \in [1, Z]$, we are given γ^z_i, with $\sum_{z=1}^{Z} \gamma^z_i = 1$. In the TIC model a propagation happens like in the IC model: when a node u first becomes active on item i, has one chance of influencing each inactive neighbor v, independently of the history thus far. The tentative succeeds with a probability that is the weighted average of the link probability w.r.t. the topic distribution of the item i:

$$p^i_{u,v} = \sum_{z=1}^{Z} \gamma^z_i p^z_{u,v}. \tag{1}$$

A TIM query $Q(\vec{\gamma}_q, k)$, takes as input an item description $\vec{\gamma}_q$ and an integer k and it requires to find the seed set $S \subseteq V$, $|S| = k$, such that the expected number of nodes adopting item q, denoted by $\sigma(S, \vec{\gamma}_q)$, is maximum:

$$Q(\vec{\gamma}_q, k) = \underset{S \subseteq V, |S|=k}{\arg\max}\ \sigma(S, \vec{\gamma}_q). \tag{2}$$

It is important to observe that a TIM query can always be processed by a standard influence maximization computation in the IC model. In fact, given the query item description, we can derive a directed probabilistic graph $G = (V, A, p)$ where the probability $p_{u,v}$ for each arc is defined as in Equation 1. This means that TIM queries maintain the same properties of standard influence maximization, thus they can exploit the standard algorithms and enjoy the usual approximation guarantees. However, as discussed previously, even the state-of-the-art algorithms for influence maximization takes from several days to weeks, on moderately sized graphs. This is clearly not suitable for real-world applications such as interactive decision support systems requiring a short response time. Therefore the goal of this paper is to build indexes to efficiently process TIM queries using precomputed information.

1.3 Challenges, contributions, and roadmap

The main challenge of using precomputed information in this setting is the enormous number of potential queries: essentially any possible item description $\vec{\gamma}_i$ which lies on the probability simplex containing all possible probability distributions with state space $[Z]$. It is also very hard to build smart indexes exploiting the graph structure, as any potential query corresponds to a different probabilistic graph.

We propose an indexing scheme, named INFLEX, (from INFluence indEX), which is motivated from the fact that similar items are likely to interest similar people, and thus are likely to have similar influence patterns. The idea is to appropriately select a set of items, and to extract their seed sets using a standard influence maximization process. Then at query time, given a query item we select a "large enough" set of neighbor index points and combine their pre-computed seed sets, by means of *rank aggregation* into a final seed set that we return as the result to TIM query.

Next, for each step, we briefly describe the associated challenge and the intuition behind the proposed solution.

Selecting the items to build the index. The number of items used to build the index governs the trade-off between accuracy and space-time efficiency. In fact, for each index point we have to run a standard influence maximization, which can be extremely time consuming, and store its seed. Another challenge is given by the space from which we have to select the index points: on one hand we want to follow the distribution observed in the catalog of items that we have available, because also new items are expected to come from the same distribution; on the other hand selecting index points directly from the catalog can be risky in the case of sparsely distributed catalog items - we might end up finding nearest neighbors which are not very similar to the query item. Our approach here is to select, for a given preprocessing budget, a reasonable number of points that can provide a good coverage of the space. This is obtained as follows. We use the catalog of available items to define, by means of a maximum likely Dirichlet distribution, the space from which we sample a large enough number of points. Then we apply K-means++ [1] to these points, and select the resulting centroids as our index points (details are provided in Section 3.1).

Fast, approximate, and unbounded nearest neighbors. At query time, given a TIM query we want to efficiently retrieve the index points which are topic-wise similar to the query item. Given that index points and query items are probability distributions over the space of topics, we adopt the Kullback-Leibler divergence as a measure of their distance. Our task is then a *similarity search* with Kullback-Leibler divergence.

Our task differs from other types of similarity search in the literature as it is not based on a pre-specified radius (*"range search"*) nor on a number of neighbors to return (*"K-NN search"*). Instead, how many points to retrieve, depends on how close the points we retrieve are to the query item. The intuition is that if we find index points extremely similar to the query item, then we can just use few of them (at an extreme, if we find exactly the query item in the index, then we can simply retrieve the associated seed set without looking for any other point). Instead, when there are no index points very close to the query point, we aggregate a larger number of them.

Another requirement for our similarity search is to be fast, for which we drop exactness: our solutions are *approximate* nearest neighbors in the sense that if we return k index points, these are not necessarily the k nearest neighbors of the query item. For this task we adopt the *Bregman ball tree* (Section 3.2) with a novel approximate nearest neighbors search procedure (Section 4.1).

Seed set aggregation. In the final step we perform rank aggregation of the seed sets[3] of the retrieved index points. The goal is to provide a final list of nodes that has the minimum Kendall-τ distance to all the seed set lists. As this problem is **NP**-hard and we aim for quick computation, we look at approximate solutions. In particular we adopt and compare Borda [3] and Copeland aggregation [7], both followed by the Local Kemenization procedure, that have been shown to be, both, fast and good in practice [25]. We enrich these two methods with a novel importance weighting scheme based on the KL-divergence of the index points from the query item: intuitively, the closer a point is to the query item, the more predominant its role will be in the aggregation (details are provided in Section 4.2).

Evaluation. The evaluation of our framework is straightforward. For a given query item, we assess the performance of the INFLEX framework in terms of accuracy and query evaluation time. For both we can compare against performing an influence maximization computation, for the given query, from scratch, as well as other smarter baselines. Our experiments on a real dataset (Section 5) show that INFLEX produces seed sets that are very close to the "best offline" ones (Kendall-τ distance generally < 0.1), while achieving an expected spread very close (NRMSE $< 3\%$) to the spread achieved by standard offline influence maximization computation, but it does so in few milliseconds instead of several hours or days of computation, thus opening the door to online influence maximization analytics.

2. OVERVIEW OF THE FRAMEWORK

In Figure 1 we already provided a very high-level view of the framework. In this section we start giving more details, opening the INFLEX box and describing its components as depicted in Figure 2. The starting point is a social graph $G = (V, A)$ where each arc $(u, v) \in A$ has associated a probability $p^z_{u,v}$ for each topic $z \in [1, Z]$ representing the strength of influence that user u exerts over user v for topic z. Moreover we have a database $\mathcal{I}$ of items, where each item i is represented by a distribution $\vec{\gamma}_i$ over the topics. Both these

[3]It is important to note that, although usually called seed "sets", these are ranked lists of nodes.

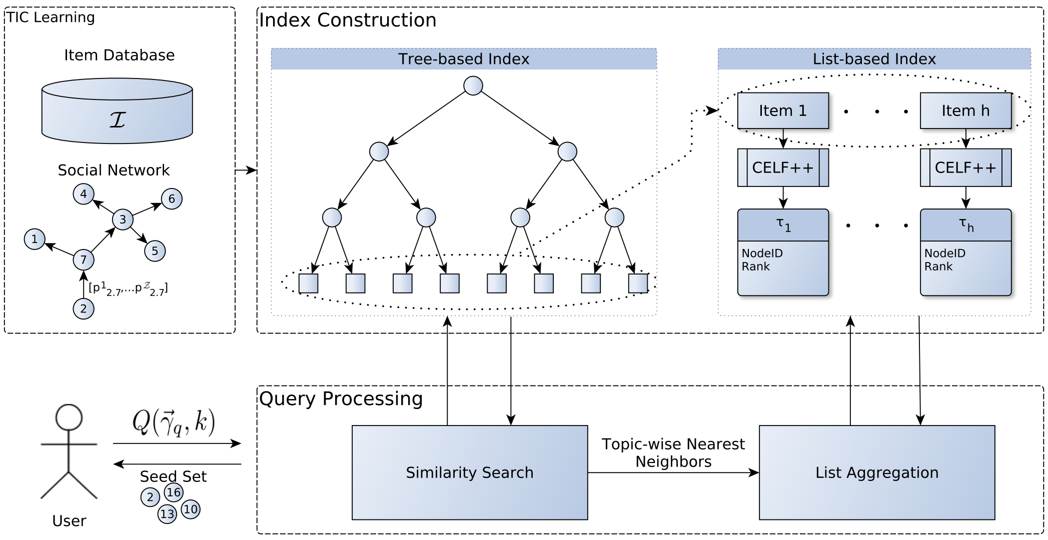


Figure 2: INFLEX framework overview.

two pieces of input (depicted in the upper-left corner of Figure 2) are jointly learnt in a pre-processing phase from a log of past propagation traces [2] (depicted in the top half of Figure 1). The database of items $\mathcal{I}$ is used to define, by means of maximum likely Dirichlet distribution, the space from which we select h index points (details are given in Section 3.1).

Let $\mathcal{H} = \{\vec{\gamma}_1, \cdots, \vec{\gamma}_h\}$ be our index points. For each $\vec{\gamma}_i \in \mathcal{H}$, and for a fixed $\ell \in \mathbb{N}$ we extract a seed set of size ℓ, or equivalently, we solve the TIM query $Q(\vec{\gamma}_i, \ell)$, by transforming it to a standard influence maximization computation over the IC model (as discussed in Section 1.2) and running the standard greedy algorithm (in particular, we use its optimization CELF++[14]). This phase is depicted in the top-right corner of Figure 2.

Now let $\tau_1, \cdots, \tau_h$ denote the index lists containing pre-computed seed sets returned for the h index points. For a given query $Q(\vec{\gamma}_q, k)$ our goal is to *(i)* retrieve the points whose topic distributions are *similar* to the topic distribution of the query item q, *(ii)* combine their pre-computed seed sets, by means of *rank aggregation* into a final seed set ${\tau_q}^*$ and return as the result to TIM query. As already discussed in the previous section, the search of index points similar to the query items is developed on top of a *Bregman ball tree* index structure (depicted in the center of Figure 2 and described in full detail in Section 3.2).

Note that the size of the seed set k requirement, can be satisfied this way even when $k > \ell$. In fact, the rank aggregation can return up to m seeds, where m is the cardinality of the union of the seed lists of all index points retrieved in the similarity search phase. By retrieving more index points, we can satisfy larger k requirements.

In the next section we present the off-line phase of the index construction. Then in Section 4 we will present the TIM query evaluation mechanism over INFLEX.

3. INFLEX CONSTRUCTION

In the TIC propagation model, each item $i \in \mathcal{I}$ is represented by a distribution over topics, $\vec{\gamma}_i$, that lies on the probability simplex $\triangle^{Z-1}$. Each topic z encodes an abstract influence pattern. The assumption is that pairwise influence probabilities between users depend on the topic. More specifically, $p^z_{u,v} \in [0,1]$ denotes the likelihood that user u will trigger the activation of user v, on topic z. Given an item i, the item-specific influence probability on each arc $(u,v) \in A$ is the dot product of the user-to-user topic dependent influence probabilities and the item's topic distribution (Equation 1). Under these assumptions, two items that exhibit a similar distribution over topics will also exhibit a similar propagation pattern, as they will enable close pairwise influence probabilities.

This observation is the core of the overall approach, as it allows us to cast the efficient processing of a TIM query as a *similarity search* problem. Intuitively, given a query $Q(\vec{\gamma}_q, k)$, we can retrieve the closest items for which the list of users to target is available, and exploit this information to provide a list of k seed nodes that can boost the adoption of q on the considered network.

The first step towards the design of the index is the formalization of the notion of similarity between two items. In this context, it is natural to instantiate the dissimilarity measure between two items as the KL-divergence between their respective topic distributions. Given two discrete distributions P and Q, the KL-divergence

$$D_{KL}(P\|Q) = \sum_i P(i) \, \log \frac{P(i)}{Q(i)}$$

quantifies the average information lost when we use Q to approximate P. Since KL is asymmetric, one must choose between the *right-sided* (the query item is the second argument) and the *left-sided* formulation, or opt for a symmetrized version that can be computed by considering the average of the sided definitions. Since our task is to retrieve the nearest-neighbors for a given query item $\vec{\gamma}_q$, the definition of dissimilarity should penalize the difference between the topic distribution of each item i and the query item, proportionally to each component γ^z_i. The dissimilarity that best suits to this setting is the right-sided KL:

$$D_{KL}(\vec{\gamma}_i \| \vec{\gamma}_q),$$

which prefers to stretch over all components γ^z_i, rather then focusing only on the highest mode of $\vec{\gamma}_q$ [22].

3.1 Selection of the index points

The topic distributions of items recorded in $\mathcal{I}$, form a *data-space* on $\triangle^{Z-1}$. This is the overall *search space* for similarity queries on topic distributions.

The first step to build INFLEX is to select a set of h index points $\mathcal{H} = \{\vec{\gamma}_1, \cdots, \vec{\gamma}_h\}$ where each $\vec{\gamma} \in \mathcal{H}$ lies on $\triangle^{Z-1}$. On one hand, we want the h points to provide a good coverage of $\triangle^{Z-1}$. On the other hand, the actual choice of the budget h depends on, both, limitations in terms of memory[4] and index construction time, due to the need of running a full influence maximization computation for each index point.

One way of selecting the index points would be to use a *space-based* approach, by selecting h items whose topic distributions are positioned *equi-distantly* on $\triangle^{Z-1}$. This would provide a fair coverage of the space. The drawback is that it disregards the available workload: in fact, the topic distribution of the items learnt from past data, might be clustered in some area of the simplex.

At the opposite extreme, we have the fully *data-driven* approach: assuming that future items will follow the same distribution of the items learnt from past data, we might select as index points, items from the catalog $\mathcal{I}$. However, this way we might end up finding nearest neighbors which are actually not very close to the query item if there are some items in the catalog whose topic distributions are sparse.

To realize a good compromise between these two indexing approaches, we resort to a sampling strategy on the simplex. By applying the *Maximum-Likelihood Dirichlet Estimation* procedure described in [20], given the topic distributions learnt from data $\Theta_{\mathcal{I}} = \{\vec{\gamma}_1, \cdots, \vec{\gamma}_{|\mathcal{I}|}\}$, we estimate the hyper-parameters $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_Z\}$ which define the Dirichlet distribution that maximizes:

$$\prod_{i \in \mathcal{I}} P(\vec{\gamma}_i|\boldsymbol{\alpha}) = \prod_{i \in \mathcal{I}} \frac{\Gamma(\sum_z \alpha_z)}{\prod_z \Gamma(\alpha_z)} \prod_z (\gamma_i^z)^{\alpha_z - 1}.$$

Then, the next step is to generate a large number of samples from *Dirichlet*($\boldsymbol{\alpha}$), identify h clusters by applying K-means++ [1], and finally use their centroids as the topic distributions of the index points.

After the selection of index items, we start building the *list-based index* that will store the seed sets returned to the preprocessing of TIM queries for the selected index items. For each item $i \in \mathcal{H}$, let G_i denote an instance of $G = (V, A, p)$ that is obtained by assigning item-specific influence probabilities, according to Eq. 1, to each arc $(u, v) \in A$. We then compute the seed set τ_i for each index item $i \in [1 : h]$ using a standard influence maximization computation.

3.2 Bregman-ball tree index

As already anticipated in the previous sections, the problem of TIM query processing can be addressed by retrieving index points which are similar to the query item. To make the similarity search phase efficient, we turn our attention to index structures to organize the points in $\mathcal{H}$.

The choice of indexing strategy for *similarity search* is naturally tied to the choice of similarity/dissimilarity function. As discussed above, INFLEX employs an information theoretic measure, the KL-divergence, which belongs to the family of Bregman divergences. This family comprises distortion measures (Squared Euclidean distance, Mahalanobis distance, Itakura-Saito distance, and KL-divergence, just to cite a few) that are defined by a strictly convex and differentiable generator function $f : \mathcal{X} \mapsto \mathbb{R}^+$ on a $d-dimensional$ convex domain $\mathcal{X}$. The Bregman divergence based on f is defined as:

$$d_f(p, q) = f(p) - f(q) - \langle \nabla f(q), p - q \rangle \tag{3}$$

where $\nabla f(x)$ is the gradient of the function $f(x)$ at point q and $\langle ., . \rangle$ denotes the dot product between two vectors. Bregman divergences are not metrics since none of them satisfies the triangle inequality and some of them fail to satisfy the symmetry property as in the case for KL-divergence. When the dissimilarity measure of interest fails to satisfy metric axioms, data structures relying on metric space axioms cannot be directly used.

For efficient similarity search with KL-divergence, we adopt the *Bregman ball tree* (bb-tree) [4, 23], a *tree-based index* structure designed to work with the family of Bregman divergences, to avoid the costly sequential scan of the database of index points with $O(Zh)$ time. Similar to its metric counterparts [5], bb-tree is built in a *top-down* fashion, by recursively partitioning the database of items to be indexed ($\mathcal{H}$), and thus defining a hierarchical space partition based on convex bodies called *Bregman balls*. A Bregman ball with a center μ and a radius R is defined as:

$$B_f(\mu, R) = \{i \in \mathcal{H} \mid d_f(i, \mu) \leq R\}. \tag{4}$$

Each node of the bb-tree corresponds to a set of database items $H_i \subseteq \mathcal{H}$ and is associated with a Bregman ball $B_f(\mu, R)$ such that $H_i \subset B_f(\mu, R)$ which covers all data points indexed in the subtree rooting at the node. Following Nielsen *et al.* [23], the tree is built from the root to the leaves, by recursively applying Bregman K-means++ at each node to generate child nodes from the parent node. The tree-branching factor is computed by applying Gaussian clustering, which allows to find the optimal number of children that avoids the overlapping of the Bregman balls of the child nodes. The tree is built in $O(h \log h)$ time for h index points.

We equip bb-tree with a novel approximate nearest neighbors search procedure, that we introduce in the next section.

4. TIM QUERY PROCESSING

In this section we present the query evaluation mechanism of the INFLEX framework. As anticipated in the previous sections it consists of two phases:

1. *similarity search* aimed at quickly retrieving a good set of index points for the given query (Section 4.1);

2. *rank aggregation* of the seed list associated to the retrieved index points (Section 4.2).

4.1 Searching for topic-wise similar items

The kind of search needed in the INFLEX framework has several peculiar requirements, that make the standard approaches, such as *range search* or *K-NN search*, unsuitable.

- The search is neither based on a pre-specified radius as in *range search*, nor on a number of neighbors as in *"K-NN search"*. Instead, how many points to retrieve is decided dynamically as the points are retrieved: if we find, in the currently visited leaf, index points similar to

[4]The cost of keeping one preprocessed index item in memory is $(Z - 1) \times$ sizeof(double) $+ \ \ell \times$ sizeof(int).

the query item, then we can stop the search. Otherwise we might need to visit more leaves.

- An extreme case is when there is an index point whose topic-distribution is identical (or extremely similar) to the query item: in that case we want to directly return the seed set of the index point, without further looking for similar points and performing rank aggregation.
- The search must be fast and can be approximate, in the sense that if it returns k results, those do not necessarily have to be the k nearest neighbors. In any case, the selected points will have a weight in the rank aggregation, proportional to their distance from the query item: so unimportant points will be treated accordingly.

Given these requirements, the similarity search in INFLEX is implemented as follows. We visit the bb-tree in *depth-first search* order, from the root to the leaf nodes, heuristically moving towards the branch whose associated Bregman ball has the center closer to the query item $\vec{\gamma}_q$, and adding the other children to a *priority queue* to ensure the early successive exploration of sub-trees that most likely contain the nearest neighbors. When we reach a leaf node, we compute the divergence of the query item from all the index points stored in the node. At this point we have three options:

1. there exists a point i in the leaf such that $D_{KL}(\vec{\gamma}_i \parallel \vec{\gamma}_q) \leq \epsilon$ for $\epsilon \approx 0$. In this case we say we have an ϵ-exact match: we stop the search and return the top-k elements in the seed set τ_i;
2. the population of points in the leaf is considered "similar enough" to the query item: we stop the search and move to the rank aggregation phase with this group of points;
3. the population of points in the leaf is not considered "similar enough" to the query item: in this case we consider the next leaf.

We have to formally define what does it mean for a group of nodes to be "similar enough". As anticipated abstractly before, this concept depends on the number of index points retrieved and on their distance from the query item. We instantiate this concept by resorting to the application of the *Anderson-Darling* test,[5] which assesses whether a sample of data comes from a population following a specific distribution. Following [23], this test has been previously applied in the phase of building the tree index, to learn the branching factor of the bb-tree by applying the G-means procedure [15]. Given a population of points which currently define a node in the bb-tree, we apply the Anderson-Darling normality test to check if, given a confidence level α, the hypothesis of normality is rejected. If this happens, the node should be split. In a similar fashion, here we check if the query item and the population of items contained in the current leaf are compatible with a normal distribution.[6] If we accept the null hypothesis that the underlying distribution is Normal, then it is likely that the population of

[5]`http://en.wikipedia.org/wiki/Anderson-Darling_test`

[6]The test is performed by projecting all the considered points in one dimension, where the application of the test is straightforward, and assuming unknown mean and unknown variance.

Algorithm 1: INFLEX similarity search

```
Input  : bb-tree $T$, query item $\vec{\gamma}_q$
Output: approximate nearest neighbors of $\vec{\gamma}_q$
$PQ \leftarrow T.root$ // init. priority queue
$NN \leftarrow \emptyset$ // init. solution set
while $PQ \neq \emptyset$ do
    $n \leftarrow$ top element in $PQ$
    while $n$ is not leaf do
        $c \leftarrow \arg\min_{c \in n.Children} D_{KL}(\mu_c \parallel \vec{\gamma}_q)$
        $PQ$.insert($n.Children \setminus \{c\}$)
        $n \leftarrow c$
    end
    if $n$ is leaf then
        if $\exists \vec{\gamma}_i \in X_n$ s.t. $D_{KL}(\vec{\gamma}_i \parallel \vec{\gamma}_q) \leq \epsilon$ then
            return $\vec{\gamma}_i$
        $NN \leftarrow NN \cup X_n$
        if similar_enough($X_n, q$) then
            return $NN$
        end
end
return $NN$
```

indexed items in the current leaf can already provide good neighbors for the query item, and hence we stop the search. The early stopping criterion based on this test achieves good performance in our framework, as we will show in Section 5.

Let B_n denote the Bregman ball $B(\mu_n, R_n)$ associated with node n, where μ_n is its center and R_n the radius, and let X_n denote the set of data points contained in node n. The overall search procedure on the bb-tree is specified in Algorithm 1.

Some implementation details are hidden in the pseudocode of Algorithm 1. The function *similar_enough*$(\cdot,\cdot)$ implements the Anderson-Darling test discussed earlier. For practical reasons the function has been implemented with the maximum number of leaves to consider. In all our experiments we keep this value equal to 5.

When we need to select the next node to explore, we can use the current solution set to produce a bound that helps us to avoid the exploration of unpromising subtrees while traversing back the tree. In particular, we use the maximum divergence of the query point from the current solution set, denoted by NN. Let δ be such divergence:

$$\delta = \max_{i \in NN} D_{KL}(\vec{\gamma}_i \parallel \vec{\gamma}_q).$$

Analogous to the triangle inequality of metric spaces, we apply the following pruning strategy. A yet unexplored node n should be visited only if the divergence of $\vec{\gamma}_q$ from the closest point in B_n is less than δ, otherwise the subtree rooted at node n can be pruned:

$$\min_{\vec{\gamma}_x \in B(\mu_n, R_n)} D_{KL}(\vec{\gamma}_x \parallel \vec{\gamma}_q) < \delta. \tag{5}$$

To test whether a node should be explored based on this strategy, we use the bisection search algorithm proposed by Cayton [4] that calculates the Bregman projection onto a Bregman ball efficiently, by using primal and dual function evaluations as the stopping criterion.

4.2 Aggregation of seed sets

We have retrieved a good-enough set of index points for the given query $Q(\vec{\gamma}_q, k)$, let us denote this set $NN(\vec{\gamma}_q)$. The next step is to aggregate their seed sets to produce the final answer for the given TIM query. Let $\mathcal{L}_q$ denote the set of the pre-computed seed lists for the index points in $NN(\vec{\gamma}_q)$, i.e., $\mathcal{L}_q = \{\tau_i \mid \vec{\gamma}_i \in NN(\vec{\gamma}_q)\}$. Our task can be nicely formalized as an optimization problem named *rank aggregation.* Intuitively, the goal is to combine the rankings provided by the pre-computed seed sets for topic-wise nearest neighbors to one *consensus* ranking which minimizes the overall disagreement. A rank aggregation function computes an aggregated ranking order which minimizes the distance to the set of orderings given as input. The ordering received as input can specify either a *full* or *partial* ranking on the objects of the domain and the techniques for rank aggregation differ slightly if they are applied to the first case or the latter.

Assume that two full ranking lists τ_1 and τ_2, defined on the same domain of n objects, are available. We can compute their distance by measuring the number of pairwise disagreements among their rankings. The Kendall-τ ($\mathcal{K}$) distance between two full lists is defined as:

$$\mathcal{K}(\tau_1, \tau_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{1}\{\tau_1(i) \prec \tau_1(j) \ \wedge \ \tau_2(j) \prec \tau_2(i)\} \quad (6)$$

where $\mathbb{1}\{.\}$ is the indicator function and $i \prec j$ is the comparison operator to denote if i is ranked ahead of j.

As we are dealing with the aggregation of lists that contain top-ℓ ranked nodes instead of complete rankings on the set of users in the network, we can employ the extension of Kendall-τ to the top-ℓ case [11]:

$$\mathcal{K}(\tau_1, \tau_2) = \sum_{\{i,j\} \in \tau_1 \cup \tau_2} \bar{K}^{(p)}_{i,j}(\tau_i, \tau_j) \quad (7)$$

where $0 \leq p \leq 1$ is a fix parameter[7] and $\bar{K}^{(p)}_{i,j}(\tau_i, \tau_j)$ is the *penalty* defined accordingly to the following four cases:

- when i and j appear in both lists: if they appear in the same order ($i \prec j$ in both lists or vice versa), then $\bar{K}^{(p)}_{i,j}(\tau_i, \tau_j) = 0$;
- when i and j both appear in one list and only i appears in the other: if $i \prec j$ in the list where they both appear, then $\bar{K}^{(p)}_{i,j}(\tau_i, \tau_j) = 0$, otherwise $\bar{K}^{(p)}_{i,j}(\tau_i, \tau_j) = 1$;
- when i appears in one list and j appears in the other list, then $\bar{K}^{(p)}_{i,j}(\tau_i, \tau_j) = 1$;
- when i and j both appear in only one of the lists, then $\bar{K}^{(p)}_{i,j}(\tau_i, \tau_j) = p$.

We normalize the Kendall-τ distance to lie in the $[0, 1]$ interval, where a distance of 0 corresponds to *identical* lists, by dividing Eq. 6 and Eq. 7 with the maximum number of possible disagreements among two full lists and among two top-ℓ lists, which is equal to $\ell(\ell - 1)/2$ and $\ell^2 + \ell(\ell - 1)p$ respectively.

The *Kemeny* optimal rank aggregation problem requires to identify the ranked list of nodes, τ_q^*, that has the minimum Kendall-τ distance to all the ranked lists $\mathcal{L}_q$:

$$\tau_q^* = \arg\min_{\tau_q} \frac{1}{|\mathcal{L}_q|} \sum_{\tau_i \in \mathcal{L}_q} K(\tau_i, \tau_q). \quad (8)$$

[7]In our calculations, we use the *neutral* approach by setting $p = 0.5$.

This optimization problem has shown to be **NP**-hard when there are at least four lists to aggregate [9]. Solutions based on Markov chains or by casting the problem as minimum-cost matching in bipartite graphs have been proposed.[8]

As efficiency is one of our main design requirements, we turn our attention to fast rank aggregation techniques, such as Borda [3] and Copeland [7] aggregation, whose result can be improved by implementing a Local Kemenization procedure.

Motivated by Social Choice Theory, rank aggregation methods treat all available rankings with equal importance. However, in our context, the aggregation should favor index lists of items who are more similar to the query item. This idea is implemented in INFLEX by incorporating importance weights into the rank aggregation.

The weighting can be further exploited to prune, for efficiency sake, lists that contribute only marginally to the final aggregation, gaining considerably in query execution time, while losing very little in terms of accuracy.

Weighted ranking aggregation techniques and a procedure for weight-based list pruning are discussed next.

Importance weights for rank aggregation. The rank aggregation module receives as input a set $\mathcal{L}_q$ of seed sets. During the aggregation, the idea is to favor the rankings entailed by index lists which correspond to the closest neighbors with respect to query item. For each $i \in NN(\vec{\gamma}_q)$ we compute a rank aggregation weight $0 \leq w_i \leq 1$, which is inversely proportional to the KL-divergence from the query item. Recall that the minimum value for KL-divergence is 0, while this measure is not bounded above. The weighting function $W : [0, \infty) \mapsto [0, 1]$ can be specified by applying a non-linear transformation of the KL-divergence values:

$$W(\vec{\gamma}_i, \vec{\gamma}_q) = \frac{e^{KL_{max}} - e^{D_{KL}(\vec{\gamma}_i \| \vec{\gamma}_q)}}{1 - e^{-KL_{max}}}, \quad (9)$$

where KL_{max} is an empirical upper bound of the KL-divergence, computed as the distance between two corners of the considered simplex and employing a smoothing factor of machine-ε value to handle zero probabilities during the computation of the KL-divergence.

Selection of nearest neighbors. Since the task of rank aggregation introduces a heavy processing burden, a careful selection of which (and how many) seed lists to consider in the aggregation is a key component towards speeding up the query evaluation phase. Therefore, we propose a procedure for the empirical selection of a subset of $\mathcal{L}_q$, based on the weighting scheme introduced above.

The idea is that by iteratively inspecting the retrieved index points, from the largest to the smallest weight, we can automatically distinguish neighbors that will contribute to the weighted rank aggregation, from neighbors whose contribution is marginal. The goal is to determine the minimum number $t \leq |\mathcal{L}_q|$, such that the top-t nearest neighbors hold the highest impact in the procedure of weighted rank aggregation. We implement this test by iteratively comparing the weight assigned to the t-th index lists with the ones assigned in the previous iteration. If the top-t nearest neighbors are equally close to the query item, then their normalized weights should tend to $\frac{1}{t}$. Let $\tilde{w}_t$ be the normalized

[8]Interested readers may refer to Schalekamp and van Zuylen [25] for a detailed comparison of rank aggregation algorithms, where algorithms having constant factor approximation or PTAS are studied in detail.

weight assigned to the t closest point, where the normalization is over all the weights up to t. We scan iteratively the set of points and stop as soon as we find a t such that:

$$\tilde{w}_t - \frac{1}{t} \geq 0.005.$$

Borda aggregation. Borda aggregation [3] is a *positional* method that corresponds to the descending order arrangement of the average Borda score for each element averaged across all ranker preferences, where Borda score for an element is the number of candidates below it in each ranker's preferences. Ordering an element by its Borda score is equivalent to ranking the vertices by increasing indegree in the corresponding weighted feedback arc set problem in tournaments and has a factor 5 approximation of the optimal Kemeny ranking [8].

Let $\mathcal{U} \subseteq V$ denote the union of users belonging to the nearest neighbors' pre-computed seed lists, i.e. $\mathcal{U} = \bigcup_{i\in[1,t]} \tau_i$. Moreover, let $\tau_i(v)$ denote the rank of the node v in the index list τ_i, and let w_i be the importance weight assigned to the i-th index list. In the case of top-ℓ lists aggregation, the weighted Borda score for each $v \in V$ can be defined as follows:

$$Borda^w(v) = \begin{cases} \sum_{i=1}^{t} w_i\,(\ell - \tau_i(v) + 1) & \text{if } v \in \mathcal{U} \\ \ell + 1 & \text{otherwise} \end{cases}$$

When $w_i = 1$, $\forall i \in [1, t]$, weighted Borda score calculation is equal to the normal Borda score calculation. For a given query $Q(\vec{\gamma}_q, k)$, the top-k nodes having the highest score are returned as output.

Copeland aggregation. Copeland aggregation [7] is a form of majority voting where the pairwise comparison among the elements in the ranked lists are taken into account. Copeland score of an element v corresponds to the number of elements v' such that v was ranked ahead, $v \prec v'$, in the majority of the lists. Copeland aggregation corresponds to the sorting of elements by non-increasing indegree on the majority tournament. The Markov Chain method (MC4) [9] is a generalization of the Copeland aggregation. For a given query $Q(\vec{\gamma}_q, k)$, we formulate the Copeland score calculation for each $v \in \mathcal{U}$ as:

$$Copeland(v) = \sum_{v' \in \mathcal{U}} \mathbb{1}\{(\sum_{j=1}^{t} \mathbb{1}\{\tau_j(v) \prec \tau_j(v')\} > \sum_{j=1}^{t} \mathbb{1}\{\tau_j(v') \prec \tau_j(v)\}\} \tag{10}$$

This can be implemented by introducing a pairwise comparison matrix $P_{v,v'}$ that stores the number of times that v precedes v' among given lists. We propose to incorporate the importance weights by promoting, in the calculation of the pairwise matrix P, those comparisons which come from index lists having greater importance weight. This weighting schema for Copeland aggregation is described in Algorithm 2. Again, for a given query $Q(\vec{\gamma}_q, k)$, the top-k nodes having the highest Copeland scores are returned as output.

Local Kemenization. Local Kemenization [9] is a greedy post-processing step which takes an initial aggregation result τ_q and computes a locally Kemeny optimal aggregation of $\{\tau_1, ..., \tau_t\}$, that is *maximally consistent* with τ_q^*. This

Algorithm 2: Weighted Copeland

```
Input  : Seed set $\tau_i$, importance weight $w_i$ $\forall$ $\vec{\gamma}_i \in NN(\vec{\gamma}_q)$
Output: Weighted Copeland scores $Copeland^w$
$\mathcal{U} \leftarrow \cup_{i\in[1,t]}\tau_i$
$P_{v,v'} \leftarrow 0$ $\forall$ $\{v, v'\} \in \mathcal{U}$
$Copeland^w \leftarrow 0$ $\forall$ $v \in \mathcal{U}$
for each $\{v, v'\} \in \mathcal{U}$ do
    for $i \leftarrow 1$ to $t$ do
        if $\tau_i(v) \prec \tau_i(v')$ then
            $P_{v,v'} \leftarrow P_{v,v'} + w_i$
        else if $\tau_i(v') \prec \tau_i(v)$ then
            $P_{v',v} \leftarrow P_{v',v} + w_i$
    end
end
for each $v \in \mathcal{U}$ do
    for each $v' \in \mathcal{U}$ do
        $Copeland^w(v) \leftarrow Copeland^w(v) + P_{v,v'}$
    end
end
```

means that no better list, in terms of lower Kendall-τ distance to all the ranked lists in input, can be achieved by just flipping an adjacent pair of elements.

We implement the procedure by an insertion sort algorithm applied on the aggregated final list. The sorting starts from the lowest ranked element in the list which is "bubbled up" as long as it is preferred by the majority of the input rankings. To apply this procedure for the weighted counterparts of Borda and Copeland aggregation, we incorporate weights into this procedure by *(i)* using weighted ranks for applying this on top of weighted Borda aggregation results, and *(ii)* using weighted pairwise comparisons on top of weighted Copeland aggregation results.

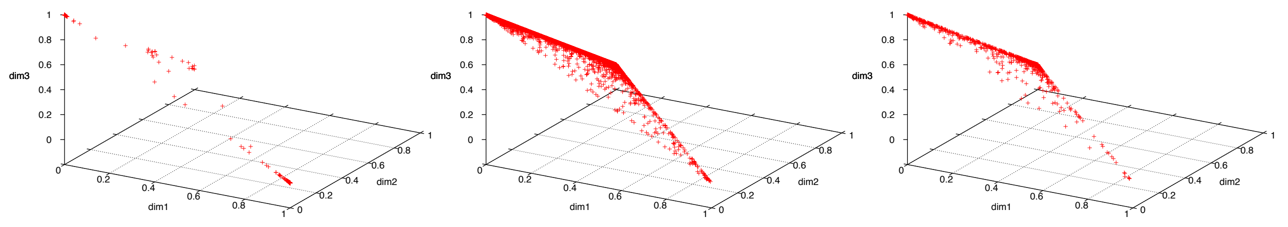
5. EXPERIMENTS

In this section we describe the experimental setup for evaluating the effectiveness and efficiency of INFLEX. The overall evaluation aims at:

- Understanding and quantifying the relationship between the distance of items in the simplex and the distance between their respective ranked list of seed nodes. In other words, to what extent the ranked seed list for an item can be used to approximate the one of its neighbors in the simplex?
- Evaluating the overall retrieval accuracy of the approximate nearest neighbors search on the bb-tree index, and the performances of the early stopping criterion based on the Anderson-Darling test.
- Comparing the performance of different rank aggregation methods and assessing the gain of the weighted versions.
- Finally, and more importantly, evaluating the accuracy of the answers provided by the overall framework and its effectiveness.

Experimental setting and dataset. Experiments were performed on a real-world dataset from Flixster:[9] a social movie web site, where users can discover new movies and share reviews and ratings with their friends.

[9]`http://www.cs.sfu.ca/~sja25/personal/datasets/`



(a) Distribution of TIC items (b) Distribution of sampled items from *Dirichlet*($\boldsymbol{\alpha}$) (c) Distribution of index items

Figure 3: Selection of index items: from the catalog (a), we learn a Dirichlet distribution that we use for sampling a large number of points (b). Index items are the centroids identified by K-Means++ (c).

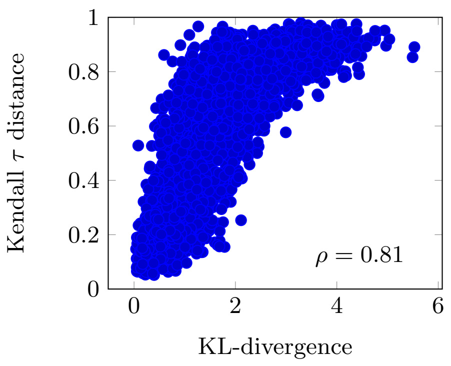


Figure 4: High correlation between the KL-divergence of items' topic distributions and the Kendall-τ distance among their seed sets.

The network is defined by roughly 30k users and 425k unidirectional social links between them, while the propagation log records the timestamp at which a user provided a rating on a particular movie, out of a catalog of 12k items. This dataset comes with the social graph and a log of past propagations (ratings on movies), and it has been widely used to test the effectiveness of social influence propagation models and influence maximization problems [13, 2]. We focus on the influence episode defined by a user v rating a movie that is later on rated by one of his friends u: in this case we see it as a potential influence of v over u. In the movie context, it is natural to assume that each item can exhibit several topics (i.e. genres) and each user may exhibit different degree of influence on different topics. We learn the topic-aware influence probabilities and the item specific topic distributions, by applying the TIC learning procedure provided in [2] and employing $Z = 10$ topics.

To evaluate the framework with respect to the aforementioned dimensions of analysis, we generated TIM queries according to, both, a data-driven and a random perspective. This differentiation allows us to study the performance of INFLEX under the assumption that query items will follow the same distributions of already indexed items, but also to assess its robustness to very diverse data distributions. To this aim, out of a total of 200 query items, half were generated by sampling from the Dirichlet distribution learnt from the item-specific distributions over topics provided by TIC learning, and the remaining were randomly generated by sampling from a uniform distribution on the simplex. As stated in the introduction of this paper, for a given query item, we assess the performance of the INFLEX framework in terms of accuracy and query evaluation time.

A detailed analysis of the experimental evaluation is provided next.

Index construction. As discussed in Sec. 3.1, the procedure for selection of items to include in the index starts with estimating the Dirichlet distribution that maximizes the likelihood of generating the item-specific distributions over topics learnt from data. To this end, we apply the *generalized* Newton iteration procedure,[10] described by Minka [20]. Then, we run Bregman K-means++ [1] over 100k samples from the Dirichlet distribution with a number of clusters equal to the number of items that we are willing to index. In this paper we use $h = 1000$. The h centroids are identified by the clustering procedure form our set of index items $\mathcal{H}$. The output of this 3-phase process is given in Fig. 3: by applying dimensionality reduction on the mapping of the $\triangle^{Z-1}$ simplex to Euclidean $\mathcal{R}^{Z-1}$ with isometric log-ratio [10], we show (a) the distribution over the topics for items in the Flixster dataset, (b) 100k samples from the Dirichlet distribution, and (c) index items.

Finally, for each item in the index we run the CELF++ algorithm for selecting their seed set. Due to extremely heavy computational burden of standard influence maximization computation,[11] we limit the seed budget for influence maximization to $\ell = 50$.

To confirm the soundness of the main assumption that motivated INFLEX, we investigate in Figure 4 the relationship between the KL-divergence among randomly selected pair of items in the index, and the Kendall-τ distance among their corresponding ranked lists. The high correlation coefficient clearly shows that the items that are close in the simplex will tend to agree on the ranking of seed nodes, while their agreement in identifying the influential nodes consistently decreases with their distance.

Retrieval accuracy of similarity search. To assess the accuracy of the similarity search on the bb-tree (Algorithm 1), we measure the recall of the search procedure in identifying the top-K true nearest neighbors, with $K \in [5, 10, 15, 20]$ on 40 randomly chosen query items. Figure 5 reports such recall for varying number of visited leaves, without using the

[10]`http://research.microsoft.com/en-us/um/people/minka/software/fastfit/`

[11]In average, the computation required 60 hours for an item, when employing 5k Monte Carlo trials with $\ell = 50$.

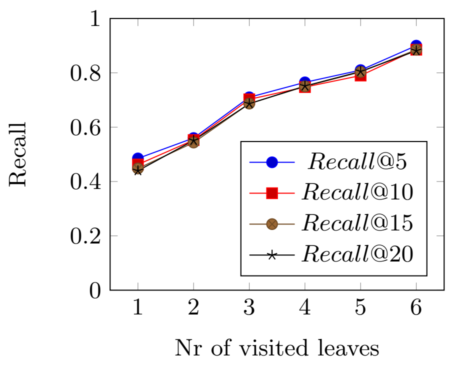


Figure 5: Retrieval accuracy in case of leaf-based search.

Anderson-Darling test as early-stopping criterion. The plot shows that when $K \leq 20$, the 80% of the top-K true nearest neighbors can be found in the first 5 visited leaves.

The accuracy of the search procedure equipped with the Anderson-Darling test is between $[0.61 - 0.63]$ for $5 \leq K \leq 20$, which is less than a 18% recall loss with respect to the case of visiting 5 leaves, using only half the KL-divergence calculations (101 vs 200 on average). We found that the average number of leaves visited when applying early-stopping criterion is 3.65. We further checked the statistical performance of leaf-by-leaf retrieval against Anderson-Darling test based early-stopping by paired t-tests ($p < 0.05$), on (i) the maximum KL-divergence observed, (ii) the retrieval recall, and (iii) the number of KL-divergence calculations. We found that our early-stopping criterion statistically works better than visiting up to 3 leaves, as it performs less KL-divergences and higher retrieval recall ($p < 0.01$). Visiting 3 or more leaves statistically provides less KL-divergences and higher retrieval recall (although with lower confidence level, $p < 0.10$). Our findings are as expected: when the number of visited leaves increases in the bb-tree, the probability of finding true nearest neighbors increases. Thus, the choice of using an early-stopping criterion is justified by a trade-off between retrieval recall and run-time, since visiting each internal node during the traversal of the tree has a computational overload of solving a convex optimization problem via Newton iterations, that is more costly than a linear-time Anderson Darling test.

Accuracy of rank aggregation. In order to assess which rank aggregation technique is better suited for INFLEX, we conduct an analysis on the accuracy of the aggregations provided by Borda and Copeland. In Table 1, we report the average Kendall-τ distance for both unweighted and weighted aggregations, obtained by processing TIM queries with different seed set sizes, while employing top-10 exact nearest neighbors search to retrieve the similar items. In general, weighted versions outperform the unweighted standard ones. Copeland^w achieves the highest accuracy (lowest Kendall-τ) and outperforms all the other techniques (paired t-tests, $p < 0.05$). We also obtained similar results with varying values for K.

TIM query evaluation. As highlighted by the previous analysis, given a list of ranked lists, their best aggregation can be achieved by employing the weighted Copeland aggregation technique. However, how to effectively select the ranked lists to pass to the aggregation module is still an open question.

INFLEX implements a fast approximate nearest neighbors search based on an early stopping criterion and a procedure for the automatic selection of ranking lists to aggregate. To evaluate the effect of the combination of these two components in retrieving good seed lists for the aggregation phase, we compare the final performance of INFLEX with the following alternatives:

- **exactKNN**: K-NN exact nearest neighbors search. This is implemented by a complete visit of the bb-tree, which provides true K nearest neighbors. The main drawback of this approach is the costly traversal time.
- **approxKNN**: K-NN approximate nearest neighbors search, realized by setting a maximum number of leaves to explore during the traversal of the bb-tree. The K nearest neighbors among the index items in the visited leaves are returned as output. This procedure provides approximate nearest neighbors as output, while it exhibits a speed up over exact search.
- **approxKNN + Sel**: K-NN approximate nearest neighbors search with automatic seed lists selection. Neighbors retrieved by the approximate nearest neighbors search are further refined by applying our procedure of automatic nearest neighbors selection. This is expected to speed up the phase of rank aggregation.
- **approxAD**: fast approximate nearest neighbors search based on the Anderson Darling test. In this case, at each leaf visited, we apply the Anderson Darling test, to decide whether or not to continue the search. This heuristic stopping criterion is expected to speed up the search in the bb-tree. Its difference from INFLEX is that we do not apply the procedure of nearest neighborsWhile on the other hand, by using the early-stopping criterion, we significantly have lower number of divergence calculations: in average ($p < 0.01$), half of the KL-divergence calculations (101 vs 200). selection.

Our experiments show that the best accuracy for K-NN based methods is achieved by employing 10 neighbors, which we assume as K in the following analysis. Figure 6 summarizes the accuracy performance of the considered methods. INFLEX outperforms in accuracy both the approximate K-NN search with automatic selection of index points and the

Table 1: Kendall-τ distance between the seed sets produced by aggregation algorithms and the ground truth computed by standard offline influence maximization computation.

Seed Set size k	*Borda*	*Borda*^{w}	*Copeland*	*Copeland*^{w}
5	0.100	0.096	0.104	**0.087**
10	0.073	0.066	0.068	**0.062**
15	0.071	0.065	0.068	**0.061**
20	0.068	0.063	0.068	**0.061**
25	0.068	0.066	0.069	**0.064**
30	0.068	0.067	0.071	**0.066**
35	0.071	**0.069**	0.072	**0.069**
40	0.074	0.073	0.075	**0.072**
45	0.079	0.076	0.077	**0.075**
50	0.081	0.080	0.079	**0.077**

fast search procedure based on the Anderson Darling test. The effectiveness of the procedure for the automatic selection of index points is witnessed by consistent gain of INFLEX over **approxAD**. The paired t-test between Kendall-τ values for INFLEX and **approxKNN** shows that there is no statistical difference between their performance in accuracy ($p < 0.01$). As expected, we see that the top performing method in terms of running time, given in Figure 7, is **approxKNN + Sel** since it applies the procedure of automatic selection of neighbors to already pre-determined number of points and reduce the computational cost of the rank aggregation procedure. INFLEX exhibits a consistent gain in running time over K-NN based methods that do not implement the automatic selection of index lists. Overall, the general framework is able to provide highly accurate estimate of seed sets in less than 30 milliseconds.

An alternative way of assessing the effectiveness of seed sets produced by INFLEX is to consider their resulting expected spread, which can be computed by running Monte Carlo simulations employing the TIC propagation model. More specifically, we compare the spread achieved by seed sets provided by INFLEX with the ones achieved by: (i) standard offline TIC influence maximization computation (offline TIC), (ii) topic-blind version of standard offline influence maximization computation that is achieved by running the TIC model with a uniform topic distribution (offline IC), and (iii) randomly selected seed sets for each item (random).

As we can see from Figure 8, the seed sets produced by INFLEX and similar aggregation-based alternatives proposed in this paper can achieve an expected spread that is very close to the one achieved by considering seed sets produced by running compute-intensive TIC influence maximization. On the other hand, seed sets identified by running topic-blind influence maximization computation perform badly, achieving less than half of the spread of the seed nodes provided by TIC influence maximization. We also provide in Table 8 the Root Mean Square Error (RMSE) and its normalized version (NRMSE) between spread values achieved by different approaches and the ones achieved by offline TIC, which is assumed as ground truth. We see that, although **approxKNN + Sel** performs better than INFLEX in terms of running time, the expected spread achieved by the solutions produced by INFLEX is much closer to the one produced by offline TIC than the solutions produced by **approxKNN + Sel** (on average 40.05 vs 98.83).

The low deviation in terms of expected spread achieved by INFLEX with respect to the ground truth, which is stable also for different choices of k as shown in Table 3, statistically confirms the accuracy as well as the robustness of the framework.

Figure 9 clearly shows that INFLEX is a good compromise as it has almost the best expected spread but using less than half the time.

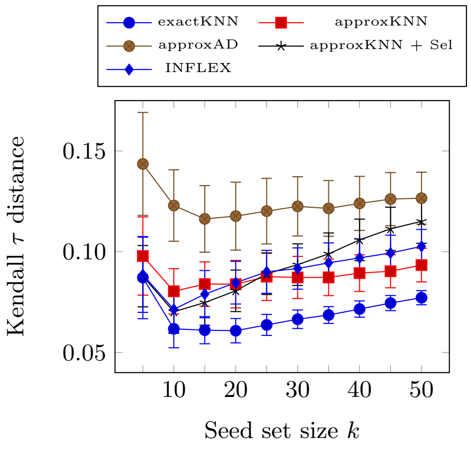


Figure 6: Accuracy comparison.

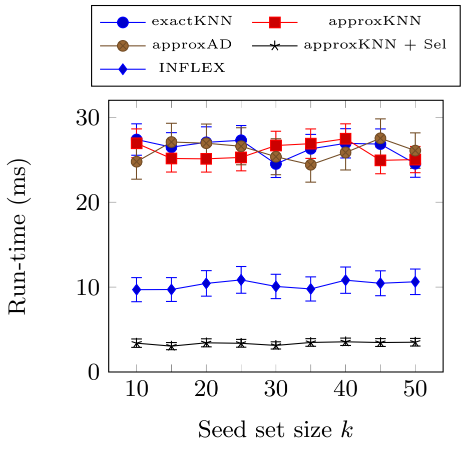


Figure 7: Run-time comparison.

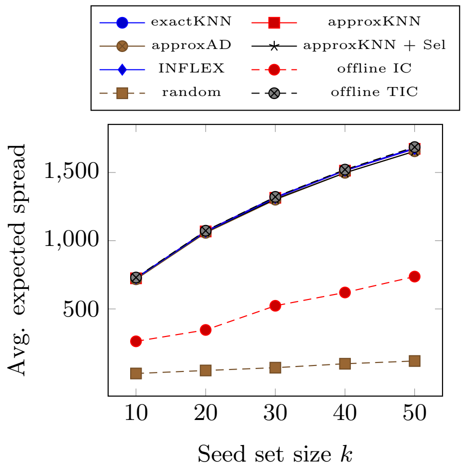


Figure 8: Expected spread comparison.

6. CONCLUSIONS AND FUTURE WORK

As a first step towards enabling social-influence online analytics in support of viral marketing decision making, in this paper we propose an efficient index for a very general type of viral marketing queries: influence maximization queries where each item is described by a distribution over a space of topics. The challenge is given by the enormous number of possible queries: essentially any point on the simplex of the topics space. Exploiting a tree-based index for similarity search in non-metric spaces, a clever approximate nearest neighbors search over the tree, and a weighted rank aggregation mechanism, our index can provide, in few milliseconds, a solution very similar (Kendall-$\tau < 0.1$) to the one produced by the offline influence maximization computation that usually takes several days, while achieving a similar ex-

Table 2: **Avg. Expected Spread of the seed sets for $k = 50$.**

Method	Exp.Spread	RMSE	NRMSE
offline TIC	1686.31 ± 60.06	-	-
exactKNN	1679.47 ± 60.12	33.23	**0.020**
INFLEX	1673.26 ± 60.80	40.05	**0.023**
approxKNN	1673.24 ± 60.84	39.24	**0.023**
approxAD	1655.30 ± 61.45	55.05	0.033
approxKNN + Sel	1655.79 ± 61.31	98.83	0.059
offline IC	737.15 ± 0.00	1020.61	1.384
random	118.47 ± 5.70	1609.30	13.583

Table 3: Accuracy of the expected spread of seeds produced by INFLEX.

k	INFLEX	offline TIC	RMSE	NRMSE
10	725.33 ± 32.24	730.39 ± 32.11	9.95	0.014
20	1066.20 ± 44.17	1073.28 ± 43.82	14.35	0.013
30	1314.08 ± 51.39	1321.32 ± 50.96	18.50	0.014
40	1513.73 ± 56.91	1521.21 ± 55.96	24.73	0.016
50	1673.26 ± 60.80	1686.31 ± 60.07	40.85	0.024

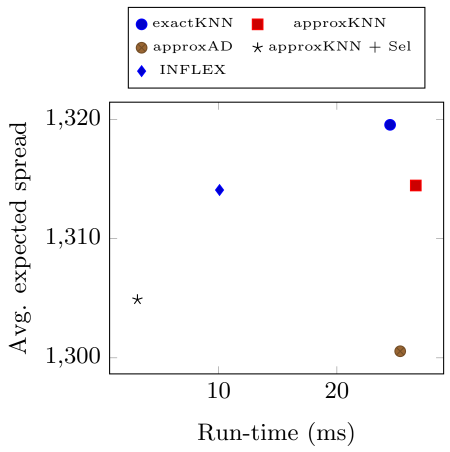


Figure 9: Run-time vs. expected spread trade-off.

pected spread as the one achieved by standard offline influence maximization computation (NRMSE $< 3\%$).

In our future work, we plan to investigate the automatic determination of the number of items to index for maintaining the accuracy of the framework, the efficient evaluation of other types of viral marketing queries (for instance, when specific market segments are targeted by the viral marketing campaign), as well as *what-if* analysis and visualization paradigms for social-influence online analytics.

Acknowledgments. This work was done during the first author's internship at Yahoo Labs Barcelona. The work is supported by MULTISENSOR project, partially funded by the European Commission, under the contract number FP7-610411 and by Grant TIN2009-14560-C03-01 of the Ministry of Science and Innovation of Spain.

7. REFERENCES

[1] A. Banerjee, S. Merugu, I. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[2] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In *ICDM 2012.*

[3] J. de Borda. Mémoire sur les élections au scrutin. 1781.

[4] L. Cayton. Fast nearest neighbor retrieval for Bregman divergences. In *ICML 2008.*

[5] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín Searching in metric spaces. In *ACM Computing Surveys (CSUR) 2001.*

[6] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD 2010.*

[7] A. Copeland. A reasonable social welfare function. Technical report, mimeo, 1951. University of Michigan, 1951.

[8] D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *SIAM 2006.*

[9] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *WWW 2001.*

[10] J. J. Egozcue, V. Pawlowsky-Glahn, G. Mateu-Figueras, and C. Barceló-Vidal. Isometric logratio transformations for compositional data analysis. *Mathematical Geology*, 35(3):279–300, 2003.

[11] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *SODA 2003.*

[12] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM 2010.*

[13] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. A data-based approach to social influence maximization. *PVLDB*, 5(1):73–84, 2011.

[14] A. Goyal, W. Lu, and L. V. Lakshmanan. CELF++: optimizing the greedy algorithm for influence maximization in social networks. In *WWW 2011.*

[15] G. Hamerly and C. Elkan. Learning the k in k-means. In *Advances in Neural Information Processing Systems*, volume 17, 2003.

[16] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD 2003.*

[17] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *KDD 2007.*

[18] X. Lin, Q. Mei, J. Han, Y. Jiang, and M. Danilevsky. The joint inference of topic diffusion and evolution in social communities. In *ICDM 2011.*

[19] L. Liu, J. Tang, J. Han, M. Jiang, and S. Yang. Mining topic-level influence in heterogeneous networks. In *CIKM 2010.*

[20] T. Minka. Estimating a dirichlet distribution, 2000.

[21] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - i. *Mathematical Programming*, 14(1):265–294, 1978.

[22] F. Nielsen and R. Nock. Sided and symmetrized Bregman centroids. *Information Theory, IEEE Transactions on*, 55(6):2882–2904, 2009.

[23] F. Nielsen, P. Piro, M. Barlaud. Tailored Bregman ball trees for effective nearest neighbors. In *Proceedings of the 25th European Workshop on Computational Geometry (EuroCG)*, 2009.

[24] K. Saito, R. Nakano, and M. Kimura. Prediction of information diffusion probabilities for independent cascade model. In *KES 2008.*

[25] F. Schalekamp and A. van Zuylen. Rank aggregation: Together we're strong. In *ALENEX 2009.*

[26] J. Tang, J. Sun, C. Wang, and Z. Yang. Social influence analysis in large-scale networks. In *KDD 2009.*