Network Bucket Testing

Lars Backstrom
Facebook, Inc.
1601 University Ave.
Palo Alto, CA 94304.

Jon Kleinberg*
Dept. of Computer Science
Cornell University
Ithaca, NY 14853.

ABSTRACT

Bucket testing, also known as A/B testing, is a practice that is widely used by on-line sites with large audiences: in a simple version of the methodology, one evaluates a new feature on the site by exposing it to a very small fraction of the total user population and measuring its effect on this exposed group. For traditional uses of this technique, uniform independent sampling of the population is often enough to produce an exposed group that can serve as a statistical proxy for the full population.

In on-line social network applications, however, one often wishes to perform a more complex test: evaluating a new social feature that will only produce an effect if a user and some number of his or her friends are exposed to it. In this case, independent uniform draws from the population on their own will be unlikely to produce a group that contains users together with their friends, and so the construction of the sample must take the network structure into account. This leads quickly to challenging combinatorial problems, since there is an inherent tension between producing enough correlation to select users and their friends, but also enough uniformity and independence that the selected group is a reasonable sample of the full population.

Here we develop an algorithmic framework for bucket testing in a network that addresses these challenges. First we describe a novel walk-based sampling method for producing samples of nodes that are internally well-connected but also approximately uniform over the population. Then we show how a collection of multiple independent subgraphs constructed this way can yield reasonable samples for testing. We demonstrate the effectiveness of our algorithms through computational experiments on large portions of the Facebook network.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous

Keywords

Social networks, Bucket testing, A/B testing, Random walks

*Supported in part by a MacArthur Foundation Fellowship, a Google Research Grant, a Yahoo! Research Alliance Grant, and NSF grants IIS-0705774, IIS-0910664, and CCF-0910940.

1. INTRODUCTION

Bucket testing, also known as A/B testing, is a practice that is widely used by on-line sites with large audiences. In a simple version of bucket testing, the site's administrators want to determine the effect of a proposed new feature on the site's users, so they expose this feature to a small randomly selected fraction of the user population and measure its effects. We call this sampled subset the *test set* for the feature. A canonical example is the scenario in which a search engine wants to evaluate a planned change to the way in which it presents its results; to decide if the new presentation increases clickthrough rates, it is shown to a small random test set, and the clickthrough rates on this test set are compared to those on the remainder of the population (who continue to see the standard presentation).

In on-line social network applications, however, one often wishes to perform a more complex test: evaluating a new social feature that will only produce an effect on a user u if both u and some number of u's friends are exposed to it. There are a range of such features, including invitations, targeted messages or ads with a social component, and pieces of information that are displayed on a user's page for the purpose of being shown to their friends. In all such cases, we may believe that a user u experiences a weak effect from the feature each time u interacts with a friend in the test set; in order to determine the effectiveness of the feature, we thus need for multiple friends of u's to belong to this test set. For simplicity, we'll assume there is a parameter $d > 0$ such that a user u in the test set is only *relevant* to the test — in other words, we'll only be able to assess the effectivness of the new feature on u — if at least d of u's friends are also in the test set.[1] We'll also assume a *budget* k such that the size of the test set should be at most k.

This is the fundamental constraint imposed by testing features in social applications — that a test set should contain individuals together with several of their friends — and it greatly complicates the problem of constructing a test set. The traditional approach, choosing a subset of k nodes independently and uniformly at random, does not work well in the network context, since when we choose a test set that is small relative to the full population, it is unlikely that the friends of a test set member will also be in the test set. Instead, we are faced with a fundamental algorithmic problem based on the following tension: we need to correlate the choices of users across the network structure, to ensure that users appear in the test set together with a sufficient number of friends, but we still need to sample approximately uniformly from the population.

[1]There are other ways of requiring u to have friends in the test set, such as assuming a probabilistic model for the effectiveness of the feature in the style of [1, 6], but the present formulation via a threshold of d is sufficient to expose the underlying issues in the problem.

Network Bucket Testing: Formulating the Problem. Although these issues are implicit in the sampling of audiences as it arises in practice, it is a problem that to our knowledge has not been precisely formulated or systematically studied. Our main contributions here are to give a concrete formulation of the network bucket testing problem, and to describe a set of algorithmic approaches to network bucket testing that improve over a sequence of increasingly sophisticated baselines that we use as bases for comparison.

First we present the formulation of the problem. We are given an n-node graph $G = (V, E)$, representing a social network on a set of users. Our goal is to estimate, as accurately as possible, the expected sum of a distribution of values across the nodes (representing their response to the feature being tested). For simplicity, we will assume the case of responses that take binary values, though our framework easily extends to more general sets of values. Thus, we assume there is a 0-1 random variable X_u associated with each node u, each with unknown and possibly distinct distributions, and our goal is to estimate the expectation of the sum $X = \sum_{u \in V} X_u$. We assume that the random variables $\{X_u : u \in V\}$ are mutually independent, although neighboring nodes in G may have distributions that are very similar to each other.[2] All our methods will produce unbiased estimators of the expectation $E[X]$, but these methods will generally differ in the variance of the estimator they produce; this variance is the objective function that we wish to minimize.

We consider methods that estimate $E[X]$ using a *test set* of nodes in G. Recall our basic premise that a node in the test set is only useful for our estimation if it also has at least d neighbors in the test set. Thus, our methods will work by first sampling a set of nodes from G, forming a *core set* C on which the random variables will be evaluated, and then adding to this a disjoint *fringe set* F designed to raise the degrees of nodes in C up to our threshold of d. Formally, then, a *test set* T consists of a pair of disjoint sets of nodes (C, F) — the core and the fringe — with the property that each $u \in C$ has at least d distinct neighbors in the union $C \cup F$. As we will see, a number of basic approaches to the construction of test sets naturally produce multisets in which elements might be repeated, so we allow T to be a multiset. In determining the size of the multiset T, we count the number of distinct nodes, and we impose the constraint that this size must be at most the budget k.

Finally, a *testing procedure* is a randomized algorithm that produces a distribution over size-k test sets $T = (C, F)$. It draws a sample (C, F) from this distribution, in which nodes u in C may appear with multiplicities $l_u > 1$. The nodes in C are the users who will be targeted by the test, and so we observe the outcomes of their random variables $\{l_u X_u : u \in C\}$. The procedure then uses a natural unbiased estimator, described in Section 3, to estimate $E[X]$ from the outcomes of these random variables. The variance of this estimator depends on the way the test set is constructed, and our goal is a testing procedure — and hence a distribution over test sets — for which the variance is as small as possible.

Designing an Algorithm for Network Bucket Testing. To develop a sense for how we can design good test sets, we discuss a sequence of general approaches, together with the trade-offs among them, that will build up to the methods we develop in the paper.

Perhaps the simplest approach is to build the core set C by making k' independent draws, for a parameter $k' < k$, uniformly and independently from the node set V. We then construct the fringe F by adding up to d nodes for each $u \in C$. The problem with this approach is that when we sample a small set C independently and uniformly from V, we will generally need d distinct nodes in the fringe for *every* $u \in C$; there are very few opportunities to use a single fringe node to raise the degree of multiple core nodes. Thus, to respect the overall size budget of k, we will need to have a core of size only about $k' = \frac{k}{d+1}$; most nodes in the test set are "wasted" in the constructon of the fringe. Since the variance of our estimate is improved by basing it on many distinct random variables, having a very small core set leads a higher variance.

To avoid wasting many nodes on the construction of the fringe, therefore, we instead pursue approaches that try to build core sets C with the property that each $u \in C$ has many neighbors in C. This way, we need to add fewer nodes to the fringe; for example, if each $u \in C$ had at least d neighbors in C, then in fact we could have a fringe F equal to the empty set. Additionally, if the set C forms a tight cluster-like structure in the graph, then we will generally be able to find fringe nodes that can raise the degrees of multiple nodes in C, again resulting in a smaller fringe.

There are two basic challenges in making this kind of approach work, and our algorithms in the paper can be viewed informally as providing methods for overcoming both of them. First, it is not enough to find a single core set C with this property of high internal connectivity; we need a distribution over such sets C with the additional property that each node of G has an approximately uniform probability of appearing in C. Without some approximate uniformity guarantee, the variance in our estimate of $E[X]$ will be very large.[3] It is computationally intractable to do this perfectly; we can show that it is NP-hard to decide whether G has a distribution over k-node sets C such that (i) each C in the support of the distribution has internal node degrees at least some fixed bound d, and (ii) each node in V has a uniform probability of appearing in a C sampled from this distribution. As a result, we will need to use heuristics to produce approximate forms of such guarantees.

Second, even if we could sample such a densely connected subset C with near-uniform probability across nodes, we would still have the following problem: nodes u and v that are connected in G may have similar distributions for X_u and X_v, and so a highly connected set C will produce a set of random variables $\{X_u : u \in C\}$ whose distributions are not representative of the full set of distributions. This *homophily bias* is another effect that increases the estimate variance. Thus we will need to build a set C with a more complex structure, consisting of multiple well-connected pieces that are globally "spread out" across the network.

Viewed in light of these difficulties, one can appreciate how effortlessly independent sampling of isolated individuals solves the problem of constructing a test set in the absence of a network constraint: in this easier case, (i) all nodes are relevant to the test, regardless of how many neighhbors they have in the test set, so there are no "wasted" nodes, and (ii) the test set is a perfect sample of the full population. In a sense, the challenge is to achieve something that works approximately as well when the network is present, dealing with the problems of wasted nodes, non-uniform sampling, and homophily bias as indicated above.

[2]This is an important point: our formulation takes into account homophily — the tendency of neighbors in a network to behave similarly — through the fact that if (u, v) is an edge of G, then X_u and X_v may be close in distribution, even though the values drawn from X_u and X_v are independent.

[3]Note: even though each node u individually should have approximately the same probability of appearing in the sample, it is clearly the case that the appearance of two neighboring nodes u and v cannot be independent. Nor would we want them to be; the goal in this style of network bucket testing is indeed to correlate the appearance of a node with its neighbors, but to do so in a way that the marginal probability of any one node's appearance is nearly uniform.

Walk-Based Methods. Our approach to constructing test sets, motivated by these issues, is to use algorithms based on random walks. For any given random walk there is a natural way to define a test set: we define the core set C to be the first h nodes visited by the walk, and then use a greedy set-cover heuristic to construct a fringe F for C, where h is selected in such a way that the size of $C \cup F$ is at most k. The advantage of using a random walk is that it automatically produces core sets C where each node in C other than the start and end of the walk has at least two neighbors in C — corresponding to the two adjacent nodes in the walk — and potentially more to the extent that the walks revisits points close to where it has been. Moreover, this internal connectivity generally makes it possible for individual fringe nodes to raise the degrees of multiple core nodes, allowing for a smaller fringe.

Since the degrees in G are not all the same, a random walk with uniform edge transition probabilities will produce a non-uniform distribution over nodes, which increases the variance of the estimate. Standard approaches to restore uniformity to such a walk involve having the walk "stall" so as to repeat low-degree nodes [2, 3, 4, 5], but this produces a very large amount of repetition in the multiset of sampled nodes, reducing the number of distinct nodes and actually providing very little improvement in variance relative to the walk with uniform transitions. Instead, we find that a significant reduction in variance is possible when we build core sets from a different random walk, with non-uniform transition probabilities computed by an iterative re-weighting scheme based on the work of Linial et al. on *matrix scaling* [7] and also related to work of Boyd et al. on the *fastest mixing Markov chain problem* [3].

Even these weighted random walks, however, tend to proceed through the network without coming close to places they've already been; as a result, while they reduce the variance well below that of standard walk-based approaches, they are still inefficient in the sense that the internal degrees of most nodes in C are only slightly above 2. We therefore introduce a further strengthening, running a random walk that operates on the *edges* of the graph rather than the vertices, and which is designed to transition from edge to edge in a way that tends to "loop back" to neighbors of nodes it has already visited. Combined with a re-weighting scheme that preserves near-uniformity in the sampling of nodes, these *triangle-closing walks* produce sets that are highly internally connected, with large internal degrees among their nodes.

Given the long history of random walks in the context of sampling [8], it is important to emphasize what random walks are accomplishing in our case and what they aren't. To begin with, random walks on large networks such as the Web graph or on-line social networks have typically been used in prior work for the problem of generating individual uniformly random nodes from a graph to which one does not have direct access [2, 4, 5]. In our case, on the other hand, we assume that the bucket testing is being performed by the administrators of the site, who can directly select individual nodes uniformly at random from their user population. Thus, random walks are not needed for this purpose, nor would they be of use if the goal were simply to perform independent sampling of nodes. Rather, random walks are used in our case to produce sets in which each node has an approximately uniform marginal probability of appearing, but which are highly connected internally; and we find that the types of random walks one needs for this purpose are quite different from those that have appeared in the prior literature on near-uniform sampling of individual nodes [2, 4, 5].

Overcoming Homophily Bias: The Bag-of-Coins Problem. Finally, we need algorithms that deal effectively with the issue of *homophily bias* discussed earlier. In particular, suppose that we run a triangle-closing random walk for a fixed number of steps, producing a set that is internally well-connected. Because the walk will tend to be concentrated in a specific portion of the network, undersampling other parts, it will lead to a set of sampled random variables $\{X_u : u \in C\}$ that have similar distributions to each other and that may not be representative of the full population. This will increase the variance, relative to a comparably-sized set of random variables whose distributions were more representative. To deal with this, we adapt our approach so that we run multiple, shorter walks with independently selected starting points. This leads to the following trade-off: longer walks produce sets with better internal connectivity, which reduces the variance by requiring fewer fringe nodes; but shorter walks produce more representative sets of random variables, which also reduces the variance.

This basic trade-off applies more generally than just to the question of walk length in our setting, and so it is useful to understand the basic optimization at a more general level. For this purpose, we abstract the trade-off in the following stylized problem.

- Suppose you are given a population of coins of different biases, and you want to estimate the overall mean bias.
- The coins are grouped into *bags*, with all the coins in a single bag having the same bias (corresponding to the fact that a single walk tends to produce nodes u with similar distributions X_u).
- There is a sublinear function $g(t)$ such that if you select a bag of coins of size t, then $g(t)$ coins are "wasted" and you can only flip $t - g(t)$. (This corresponds to the fact that walks of different length result in different numbers of nodes wasted on the construction of the fringe.)

If you are allowed to choose bags of a fixed size totaling k coins, what is the best bag size to pick if you want to minimize the variance in your estimate? In a simple form, this captures the trade-off we experience in our walks. Choosing a few large bags takes advantage of "economies of scale" (fewer coins are wasted), but there is extensive sharing of biases among the coins. Choosing many small bags allows for a good mix of biases, but it wastes many coins.

We will see in Section 2 that this bag-of-coins model has an appealing solution in which an interior optimum naturally emerges for the best bag size — balanced between the extremes of bags that are too small and those that are too large. We find a surprisingly accurate reflection of this interior optimum when evaluating our algorithms on real networks; the variance is minimized by dividing the walk into independent segments of length h, for an h balanced between walks that are too long (visiting too many nodes of similar distribution) and too short (wasting too many nodes on the fringe).

Evaluating our Methods. This completes the high-level description of our full method for constructing test sets. We build the sets by performing triangle-closing random walks, with transition weights computed in a way that gives each node an approximately uniform probability of being visited by the walk. We restart the walk from a randomly chosen node every h steps, for a value of h chosen to optimize the variance of the resulting estimate.

To evaluate our methods, and to supplement the theoretical arguments justifying the basic ingredients of the walks, we perform experiments in which we estimate random values distributed across nodes in a large subgraph of the Facebook network. We find that each of the components in our approach leads to a significant improvement in the quality of the estimate: re-weighted walks are more effective than traditional "uniformized" walks with stalling; triangle-closing walks are more effective still because they create high internal connectivity in the sampled set; and restarting the walk at optimally timed intervals reduces the variance by creating random variables that better represent the full population.

2. A MODEL OF OPTIMAL WALK LENGTH

We begin by analyzing the stylized probabilistic model described in Section 1, using bags of coins of different sizes to abstractly capture the trade-off between taking a few long walks that are more efficient in their use of fringe nodes, and taking many short walks that better represent the set of possible distributions in the population.

In our simple model, there is a large universe of coins of types $1, 2, \ldots, m$. Coins of type i have a probability p_i of coming up "heads," and an f_i fraction of all coins are of type i, where $\sum_{i=1}^{m} f_i = 1$. We do not know the values $\{p_i\}$ or $\{f_i\}$, but we would like to estimate the probability of heads, $\sum_{i=1}^{m} f_i p_i$, by flipping a set of coins and using the observed number of heads as an estimate.

In our model, we have a fixed budget k, and we can request s bags of coins of size t each, where we can choose s and t subject to the constraint $st \leq k$. When we receive a bag of coins, they all have the same type; type is i with probability f_i. We cannot flip all t coins in the bag, however; instead, for a function $g(\cdot)$, we can flip $\widetilde{t} = t - g(t)$ coins. Let X_j be a random variable equal to the number of heads observed among flips from the j^{th} bag we select, and let $X = \sum_{j=1}^{s} X_j$. We want to choose t (and hence s) so that

$$\Pr\left[(X > (1+\varepsilon)E\left[X\right]) \cup (X < (1+\varepsilon)E\left[X\right])\right]$$

is minimized. Since this probability is approximately controlled by $\frac{\sqrt{\text{Var}[X]}}{E[X]}$, we will seek t to minimize this expression.

As noted in Section 1, this is not exactly the problem we face in the graph G, but it is a stylized abstraction of it; the function $g(t)$ roughly corresponds to the "waste" of nodes due to the fringe around a walk segment of length t, and the shared type of all coins in a bag roughly corresponds to the similarity in distributions of nodes on the same walk segment. In Section 4, we will see that this analogy is close enough to permit strong numerical agreement.

Analysis for Two Types. We describe how to analyze this bag-of-coins model in the simple case when there are two types of coins, of bias $p_1 = a$ and $p_2 = b > a$, and of equal prevalence $f_1 = f_2 = \frac{1}{2}$ in the population. We then summarize how to generalize the analysis to an arbitrary set of types.

We begin the analysis with a basic calculation about variances, which is useful in the analysis. Suppose we have a random variable Z defined in terms of two other random variables Z_1 and Z_2 as follows. With probability $\frac{1}{2}$ we draw a value from Z_1, and with probability $\frac{1}{2}$ we draw a value from Z_2. Let $\mathcal{E}_1$ be the event that we draw from Z_1 in determining Z, and $\mathcal{E}_2$ be the event that we draw from Z_2. Then we have

$$\begin{aligned}
\text{Var}\left[Z\right] &= E\left[Z^2\right] - E\left[Z\right]^2 \\
&= E\left[Z^2 \mid \mathcal{E}_1\right] \cdot \Pr\left[\mathcal{E}_1\right] + E\left[Z^2 \mid \mathcal{E}_2\right] \cdot \Pr\left[\mathcal{E}_2\right] \\
&\quad -\frac{1}{4}(E\left[Z_1\right] + E\left[Z_2\right])^2 \\
&= \frac{1}{2}(E\left[Z_1^2\right] + E\left[Z_2^2\right]) - \frac{1}{4}(E\left[Z_1\right] + E\left[Z_2\right])^2 \\
&= \frac{1}{2}(E\left[Z_1^2\right] + E\left[Z_2^2\right]) \\
&\quad -\frac{1}{4}(E\left[Z_1\right])^2 + 2E\left[Z_1\right]E\left[Z_2\right] + E\left[Z_2\right]^2) \\
&= \frac{1}{2}(E\left[Z_1^2\right] - E\left[Z_1\right])^2) + \frac{1}{2}(E\left[Z_2^2\right] - E\left[Z_2\right])^2) \\
&\quad +\frac{1}{4}(E\left[Z_1\right])^2 - 2E\left[Z_1\right]E\left[Z_2\right] + E\left[Z_2\right]^2) \\
&= \frac{1}{2}\text{Var}\left[Z_1\right] + \frac{1}{2}\text{Var}\left[Z_2\right] + \left(\frac{E\left[Z_2\right] - E\left[Z_1\right]}{2}\right)^2
\end{aligned}$$

Now, recall that X_j is the number of heads observed among coin flips from the j^{th} bag we select. To compute the variance of X_j, we can set $Z = X_j$ and use our formula for $\text{Var}\left[Z\right]$ from above, with Z_1 corresponding to the case in which draw $\widetilde{t}$ coins of mean a, and Z_2 corresponding to the case in which draw $\widetilde{t}$ coins of mean b. Also, let $c = (a+b)/2$ and $d = (a-b)/2$. We have $E\left[Z_1\right] = \widetilde{t}a$, $E\left[Z_2\right] = \widetilde{t}b$, $\text{Var}\left[Z_1\right] = \widetilde{t}a(1-a)$ and $\text{Var}\left[Z_2\right] = \widetilde{t}b(1-b)$, so

$$\begin{aligned}
\frac{\frac{1}{2}\text{Var}\left[Z_1\right] + \frac{1}{2}\text{Var}\left[Z_2\right]}{\widetilde{t}} &= c - \frac{1}{2}(a^2 + b^2) \\
&= c - \frac{1}{2}((c-d)^2 + (c+d)^2) \\
&= c - (c^2 + d^2) \\
&= c(1-c) - d^2.
\end{aligned}$$

Writing $v = c(1-c)$, we have

$$\text{Var}\left[X_j\right] = \text{Var}\left[Z\right] = \widetilde{t}v - \widetilde{t}d^2 + \widetilde{t}^2 d^2 = \widetilde{t}v + (\widetilde{t}^2 - \widetilde{t})d^2.$$

Finally,

$$\begin{aligned}
\text{Var}\left[X\right] &= \text{Var}\left[\sum_{j=1}^{s} X_j\right] \\
&= s(\widetilde{t}v + (\widetilde{t}^2 - \widetilde{t})d^2) \\
&= \frac{k\widetilde{t}(v + (\widetilde{t} - 1)d^2)}{t}
\end{aligned}$$

and

$$E\left[X\right] = s\widetilde{t}c = \frac{k\widetilde{t}c}{t}.$$

(For simplicity in the analysis, and to permit closed-form expressions, in the above we use an approximation where we drop integer constraints on s and t. Our numerical experiments indicate that this does not cause any significant source of error.)

Minimizing $\frac{\sqrt{\text{Var}\left[X\right]}}{E\left[X\right]}$ is the same as minimizing $\frac{\text{Var}\left[X\right]}{E\left[X\right]^2}$, and we can write the latter as

$$f(t) = \frac{t}{k\widetilde{t}c^2}\left[v + \left(\widetilde{t} - 1\right)d^2\right] = \frac{t}{kc^2}\left[\frac{v}{\widetilde{t}} + \left(1 - \frac{1}{\widetilde{t}}\right)d^2\right].$$

Now the point is that for functions $g(t)$ that are monotone increasing and grow as $o(t)$, we should typically expect to see this function $f(t)$ achieve an interior optimum strictly between 1 and k. If $g(t)$ is differentiable, then we can investigate the value of this optimal t by considering the equation $f'(t) = 0$.

As an illustration of this, let's consider the simple case in which $g(t) = r$, so that $\widetilde{t} = t - r$. For notational simplicity, we also write $\widetilde{v} = v - d^2$, which we note is positive by the definition of c and d.

For this choice of $g(t)$, we have

$$f'(t) = \frac{1}{kc^2}\left[\frac{d^2 t}{t-r} - \frac{r}{(t-r)^2}(\widetilde{v} + d^2(t-r))\right].$$

Setting $f'(t) = 0$, we get (when $t > r$)

$$d^2 t = \frac{r\widetilde{v}}{t-r} + rd^2.$$

It follows that

$$(t-r)^2 = \frac{r\widetilde{v}}{d^2}$$

and hence

$$t = r + \frac{\sqrt{\widetilde{v}}}{d} \cdot \sqrt{r} = r + \sqrt{\frac{v}{d^2} - 1} \cdot \sqrt{r}.$$

The conclusion is somewhat surprising: when $g(t)$ is a constant, the optimal grouping strategy chooses to waste asymptotically almost all of the samples so as to produce a relatively large number of distinct groups — that is, it loses r samples in order to produce a group with $O(\sqrt{r})$ usable samples.

Now we consider what happens when we carry out the optimization for more general functions $g(t)$. We have

$$f(t) = \frac{t}{kc^2}\left[\frac{v}{\widetilde{t}} + \left(1 - \frac{1}{\widetilde{t}}\right)d^2\right]$$

and

$$f'(t) = \frac{1}{kc^2}\left[\frac{\widetilde{v}}{\widetilde{t}} + d^2 - \frac{t\widetilde{t}'\widetilde{v}}{\widetilde{t}^2}\right].$$

Setting $f'(t) = 0$ and assuming $\widetilde{t} > 0$, we have

$$\widetilde{v} + \widetilde{t}d^2 = \frac{t\widetilde{t}'\widetilde{v}}{\widetilde{t}}.$$

Now substituting in $\widetilde{t} = t - g(t)$

$$\widetilde{v} + (t - g(t))d^2 = \frac{t(1 - g'(t))\widetilde{v}}{t - g(t)}.$$

We can rearrange to find

$$\frac{g(t) - tg'(t)}{(t - g(t))^2} = \frac{d^2}{\widetilde{v}}.$$

If we assume that $g(t) = \beta t + h(t)$, with $h'(t)$ going to 0 (as in $g(t) = \beta t + \gamma t^\delta$ with $\delta < 1$, for example), then we can substitute $t = g(t) + C\sqrt{g(t)}$ and find that

$$\begin{aligned}
\frac{d^2}{\widetilde{v}} &= \frac{g(t) - (g(t) + C\sqrt{g(t)})g'(t)}{C^2 g(t)} \\
&= \frac{1 - g'(t)}{C^2} - \frac{g'(t)}{C\sqrt{g(t)}} \\
&= \frac{1 - \beta - h'(t)}{C^2} - \frac{\beta + h'(t)}{C\sqrt{g(t)}} \\
&\approx \frac{1 - \beta}{C^2}
\end{aligned}$$

When $h'(t)$ and $\frac{\beta + h'(t)}{C\sqrt{g(t)}}$ are both converging to 0, we find that $C \approx \frac{\sqrt{\widetilde{v}}}{d\sqrt{1-\beta}}$, and that the optimal value of t occurs when $t \approx g(t) + \frac{\sqrt{\widetilde{v}}}{d\sqrt{1-\beta}}\sqrt{g(t)}$. This is a natural generalization of the earlier special case when $g(t) = r$, and with the same qualitative conclusion.

Larger Sets of Types. With an arbitrary set of types, the analogue of our calculation for $\mathrm{Var}[Z]$ can be derived from the *Law of Total Variance*, which tells us that for random variables Z and Y, we have $\mathrm{Var}[Z] = E[\mathrm{Var}[Z|Y]] + \mathrm{Var}[E[Z|Y]]$. We denote the first of the terms in this sum by $E(V)$, and the second term by $V(E)$.

Thus, when we sample s buckets of cost t gaining $\widetilde{t}$ coins each, we get a variance of

$$\mathrm{Var}[X] = \frac{k}{t}(\widetilde{t}E(V) + \widetilde{t}^2 V(E))$$

This gives us

$$f(t) = \frac{\mathrm{Var}[X]}{E[X]^2} = \frac{t(E(V) + \widetilde{t}V(E))}{k\widetilde{t}E[X]^2}$$

and hence

$$\begin{aligned}
k\widetilde{t}^2 E[X]^2 f'(t) &= \widetilde{t}E(V) + \widetilde{t}^2 V(E) + t\widetilde{t}\widetilde{t}'V(E) \\
&\quad -t\widetilde{t}'E(V) - t\widetilde{t}\widetilde{t}'V(E) \\
&= (t - g(t))E(V) + (t - g(t))^2 V(E) \\
&\quad -t(1 - g'(t))E(V) \\
&= -g(t)E(V) + (t - g(t))^2 V(E) + tg'(t)E(V)
\end{aligned}$$

Setting the derivative to 0, we find that

$$g(t)E(V) = (t - g(t))^2 V(E) + tg'(t)E(V)$$

This provides an equation that can be solved for the optimal t. The form of the solution will depend on the specific parameters, but we can check for example that this equation includes our previous two-type special case with $g(t)$ equal to a constant r. In this case we have $E(V) = c(1 - c) - d^2 = \widetilde{v}$ and $V(E) = d^2$, and hence $r\widetilde{v} = (t - r)^2 d^2$ as before.

From bags of coins to core sets. For assembling a core set from random walk segments of length t, this bag-of-coins model thus provides a way of reasoning about how the variance depends on the choice of t, and hence how to find an optimal value for t. However, the composition of the bag itself — the analogue of the set of nodes sampled by each walk segment — is treated in a "black box" fashion by this simple model. We can see from the expression for $\mathrm{Var}[Z]$ at the beginning of this section that reducing the variance of the overall estimate involves reducing the variance within the bag of coins selected, but the model itself does not provide guidance on how to sample nodes to accomplish this. Accordingly, in the next section we consider how to perform a random walk of a given length so as to select a set of nodes for which the variance of the resulting estimate is low.

3. ALGORITHMS

Given the abstract framework developed when considering the bags-of-coins problem, we now move on to develop algorithms which in essence, construct the bags of coins. Now though, each coin reprsents a vertex in the graph, and each bag respresents a multiset of vertices, which are related in some way by the network. Similar to the bags-of-coins problem, we have a fixed budget of nodes who we may expose to the test condition that we are evaluating. However, we can only evaluate the feature on nodes who have at least d neighbors also exposed to the test condition.

Recall the tradeoff between the two extremes in the introduction; at one extreme we could sample $\frac{k}{d+1}$ isolated nodes, spending the rest of our budget on a fringe to bring them up to a minimum degree of d, while at the other extreme we could sample a single group of k well-connected nodes, avoiding the use of many fringe nodes but at the cost of potential non-uniformity and overlap in node selection. Our algorithms will walk a middle ground between these two extremes. We strive to come up with algorithms which are efficient in the sense that they test on a large fraction of the budget, but also have enough uniformity and independence to provide a low-variance estimate of X.

One general way to do this it to attempt to sample many small connected groups of nodes (in the style of our bags-of-coins model). The hope here is that if we sample a connected group of nodes, we will be able to choose their neighors in such a way that we cover them more efficiently, and waste less than d additional nodes per core-set node. To take the simplest example, we could sample a pair of connected nodes u and v. In this case, we start with both u and w having degree 1. We could then add all w such that $(u, w) \in E$

and $(v, w) \in E$. If there were j such w, then we could add those j nodes, which each increase the degrees of both v and w, along with $2(d - j - 1)$ other nodes that only each increase the degree of one of v or w. This would bring the degrees of u and v up to d, and so we could evaluate the test on two nodes at a cost of $2d - j$, which is more efficient than evaluating on one node at a cost of $d + 1$.

There are two potential problems with this approach. The first stems from the fact that u and v are likely to be related, but this is precisely the issue addressed in Section 2, and we have seen that under the right conditions, the increase in variance due to this relationship is more than offset by the decrease in variance due to increased efficiency. The other problem is that this method does not sample nodes uniformly, and if we were to naively take the observed average, we would not have an unbiased estimate.

More specifically, if we were to sample a uniform edge, we would expect a node u to appear with probability proportional to its degree. In order to make this sampling method an unbiased estimator of X, we need to take into account the expected observation frequency of each node imparted by our sampling strategy. The correct way to do this is simply to divide each observation by its expected frequency, p_i. Thus, if our budget allowed us to sample t edges u_t, v_t, along with their supporting $2d - j_t$ neighbors, we would estimate X as

$$C \sum_i \left(\frac{X_{u_i}}{d_{u_i}} + \frac{X_{v_i}}{d_{v_i}} \right)$$

where C is a normalizing constant that depends on $|V|$, t, and $|E|$.

This brings us to a more general approach, which we will use in two of our baseline methods, along with the three algorithms we develop. A single edge can be thought of as a walk of length 1, which has some known bias to select some nodes more frequently than others. We can generalize this to walks of length greater than 1 which may also be biased to visit some nodes more frequently than others, but in a way that we can correct for. Thus, our approach is to take a number of random walks according to some strategy, which imparts an expected visitation distribution on the nodes. We will measure X for all of the nodes visited, and then use a correction similar to what we outlined above to obtain an unbiased estimate of X. Whereas sampling a single edge causes a node to appear with probability proportional to its degree, various walk strategies will cause nodes to occur with other distributions.

In general, we will estimate X based on t independent random walks. In these t random walks we will encounter a set of nodes U, and an individual node u will appear l_u times, while, given a walk strategy, it was expected to appear p_u times. Our final estimate of X will thus be

$$\frac{|V|}{|U|} \sum_{u \in U} \frac{X_u l_u}{p_u}$$

For each of the t walks, we need to increase the degrees of the core test nodes to at least d. To do this, we perform a greedy covering algorithm after each walk. If the set of nodes in the walk is U, and $U' \subseteq U$ of them have degree less than d, then we repeatedly select a node w such that $|N(w) \cap U'|$ is maximized; that is we select w to increase the degree of as many nodes in U' as possible. In this way we obtain the set of *core* nodes which are found in the walk, and a set of supporting *fringe* nodes whose role is simply to increase the degrees of the core nodes to d. We will only measure X on the core nodes.

In the rest of this section, we introduce three baselines to compare against, and then develop algorithms which outperform all of these baselines. A guiding motivation underlying these algorithms, even the more elaborate ones, is that they should all be able to construct the distributions needed for the random walk by iterative methods that update weights through local computations across nodes and edges; in this way, they can be made to scale up to graphs that are the size of large on-line social networks.

3.1 Baselines

We first describe the baseline methods; the third one is in fact often used by state-of-the-art methods for node sampling, but in the context of our problem it will be a point relative to which our subsequent algorithms will improve significantly.

1. Isolated Sampling. This strategy samples nodes uniformly, and for each sampled node we pick d of its neighbors. To estimate the entire population X, we simply multiply the outcome in our sample by $\frac{|V|(d+1)}{k}$. This makes the overall variance of the method scale as $\mathrm{Var}[X] \frac{|V|(d+1)}{k}$.

2. Unweighted Random Walk. An unweighted random walk starts at a random node, and at each step advances to a neighbor of the current node. The stationary probability of a node u in such a walk — the long-run probability of being at a u after many hops – is proportional to the degree d_u. However, in this version we start at a node selected uniformly at random. While this means that p_u does not have a simple closed form, we can compute it as $\sum_l p_u^l$, where p_u^l is the probability of being at u after l hops in the walk. p_u^l can be computed with dynamic programming as $p_u^0 = \frac{1}{|V|}$ and $p_u^{l+1} = \sum_{(v,u) \in E} \frac{p_v^l}{d_v}$. We note that this computation can be done in $O(|E|L)$, and is readily distributed and parallelizable.

3. Metropolis Sampling Random Walk. One way to avoid the bias towards high-degree nodes is by using Metropolis sampling when taking the random walk [3, 4]. As in the unweighted random walk, we will select a neighbor uniformly at random from amongst all neighbors. However, if we are currently at u and we select v as the next hop, but $d_v > d_u$ then we will, instead of hopping, stay at u with probability $\frac{d_u}{d_v}$. This sampling method ensures that all nodes are equally likely at every step of the walk.

However, the obvious drawback here is that the walks tend to have fewer unique nodes in them. One might think that we could simply run this algorithm until l unique nodes had been visited for some l, but this introduces the bias that p is no longer uniform (indeed, it would be a sort of interpolation toward the unweighted walk just discussed). Thus, we will take walks of length l with the expectation that many of the nodes will have multiplicity $l_u > 1$.

3.2 Our methods

Algorithm 1. Weighted Walks. The unweighted random walk had the nice property that we rarely visited the same node more than once, giving nodes low multiplicity, which is good insofar as the variance of the estimate is concerned. However, the bias it imparted increased the variance. On the other hand, Metropolis sampling had no bias that needed to be corrected, but the high multiplicity is bad for the variance. In this approach we attempt to achieve the best of these two methods. We will take a random walk, with no stalling, but we will weight the edges in such a way that this does not bias towards high-degree nodes.

In the ideal case, we would end up with a distribution on the outgoing edge weights $0 \leq w_{u,v}$, such that $\sum_u w_{u,v} = 1$ and $\sum_v w_{u,v} = 1$. Doing so would ensure that we were at each node with equal probability at each step in a random walk, but we hope it

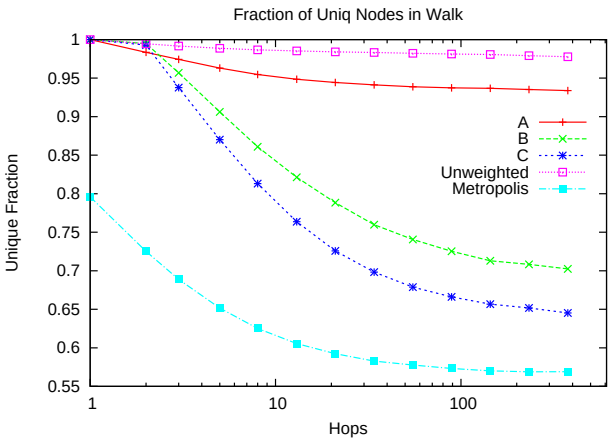

Figure 1: Fraction of nodes that are unique in a walk.

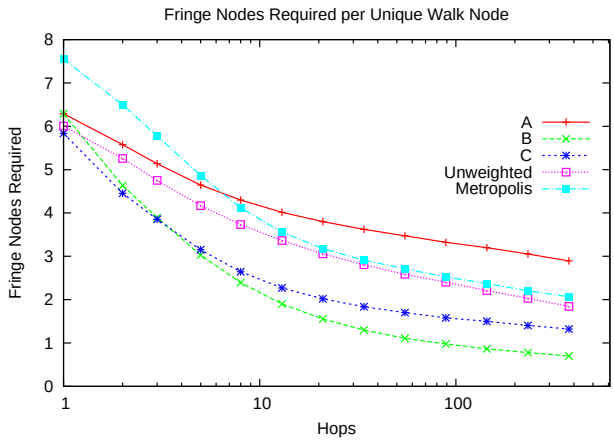

Figure 2: Number of nodes required to give everyone degree at least 10.

would not have the multiplicity drawback of Metropolis sampling. Finding such a set of weights is not possible for all graphs, but if we can come close, we can correct for any non-uniformity by using the appropriate distribution p.

One approach to finding a set of weights is to view the above as a set of linear constraints, and use standard methods for finding a solution. However, this becomes infeasible for large graphs with millions or billions of edges.

Instead, we take an iterative approach which converges to such a set of weights if one exists. The approach is one that is employed by Linial et al. [7] for a problem they call *matrix scaling*, and is based on earlier work of Sinkhorn [9]. In each iteration, we perform the following two steps:

$$1:\ w'_{u,v} \leftarrow \frac{w_{u,v}}{\sum_u w_{u,v}}$$

$$2:\ w_{u,v} \leftarrow \frac{w'_{u,v}}{\sum_v w'_{u,v}}$$

In other words, we repeatedly normalize the incoming weight of each node so that it sums to 1, and then normalize the outgoing weight of each node so that it sums to 1.

We find that this algorithm converges rapidly in practice, and after running this algorithm for a number of rounds (100 in our experiments) the probability distribution over nodes after t steps is very nearly uniform.

Algorithm 2. Weighted Triangle-Closing Walks. While the weighted walks achieve near-uniformity and low multiplicity, they are still not very efficient. Though internal nodes other than endpoints have degree at least 2 after the walk (ignoring the occasional repitition), they still tend to need many nodes added to the fringe to bring the induced degree of the core nodes up to d.

One approach to correcting this is to attempt to make the walks more compact, and a simple way to do this is to bias it towards closing triangles. More specifically, when we are at node u, we remember the previous node in the walk s. If $(s, v) \in E$, we bias the walk towards v. The hope is that by doing this we will end up with walks that have higher internal degrees among the nodes they visit and which are more compact, requiring fewer additional fringe nodes, and hence making more efficient usage of our budget.

In detail, we learn the weights for this algorithm in the same way we did for Algorithm 1. We start the walk at a node u_0 chosen uniformly at random, and take the first hop to u_1 as in Algorithm 1. Beyond that we use the current node u_i and the previous node u_{i-1} to weight u_i's neighbors. A node v is chosen with probability proportional to $w_{u_i,v}$ if $(u_{i-1}, v) \notin E$ and with probability proportional to $\alpha w_{u_i,v}$ if $(u_{i-1}, v) \in E$, for some constant α. In other words, steps that close triangles get multiplicitive boosts to their weight. Another way of thinking about this is that it is a Markov process where the state is now an edge instead of a node, and the transition weights between states are computed based on the weights w.

It is important to note that even if the weights learned in Algorithm 1 gave us uniformity over nodes, this bias toward closing triangles breaks that. Thus, to make this algorithm an unbiased estimator, we must compute p_u for all nodes. While this is bad in some sense, since non-uniformity increases variance, a walk where many hops close triangles will give interior nodes larger initial degrees, before the fringe is added. (For example, if every step closed a triangle and all nodes were distinct, the degrees of all interior nodes would be at least 4.) The process will also tend to give sets of nodes that are more compact and hence can be supported by a smaller set of fringe nodes. We will find that this trade-off between non-uniformity and compactness works out in favor of this triangle-closing rule.

Algorithm 3. Uniformized Triangle-Closing Walks. The triangle walk is appealing in that it creates compact sets of nodes with high internal degrees. However, it is clearly biased towards high-degree nodes. Could we come up with a weighting on the edges such that the triangle closing walk was more nearly uniform? In the weighted walk, we were able to correct for the bias towards high-degree nodes through an iterative approach, and we now show that it is possible to do the same thing here for the triangle-closing walk.

An intial attempt at this would be to apply the same approach as in the weighted algorithm. In this case it would work slightly differently, however, as we would like to assume that we are at a random node, and then take two hops, and somehow adjust all the weights so that the walk becomes more nearly uniform. We have a problem here though: the walk is biased to close triangles, and so we can not simply take two hops from a random node, as the first hop depends on the predecessor.

To account for that, we will correct our iterative procedure to learn a distribution over predecessor nodes as it is updating the weights on the edges. As before, we imagine that there are weights $w_{u,v}$ on all the edges, and at step i in a walk $u_0, u_1, ..., u_i$, we select node v with probability proportional to to $(1+\alpha\mathbf{1}((u_{i-1}, v) \in$

$E))w_{u_i,v}$, where $\mathbf{1}((u_{i-1},v) \in E))$ is 1 if $u_{i-1}, v \in E$ and 0 otherwise. This weights triangle-closing steps by a factor of α. For notational simplicity, we will write the probability of a walk going from y to z, given that it came from x as

$$w_{x,y,z} = \frac{(1+\alpha\mathbf{1}((x,z) \in E))w_{y,z}}{\sum_{z'}(1+\alpha\mathbf{1}((y,z') \in E))w_{y,z'}}$$

.

Given this process, there is some stationary distribution of predecssor nodes. That is, if we walk for many hops and are at currently at node u_i, what is the distribution over u_{i-1}. We will define this probability distribution $q_{v,u} = P(u_{i-1} = u | u_i = v)$. We next develop an algorithm of the same flavor as the weighted walk algorithm. Assume that we are at each node u with probability $\frac{1}{|V|}$, and that the predecessor of u is s with probability $q_{u,s}$. We can now compute the probability distribution over nodes in the next two hops.

If we are currently at u, the probability that the next hop will take us to v is $p_v(u) = \sum_s q_{u,s}w_{s,u,v}$. Having computed the probability distribution over this first hop, we can go one step further, computing the probability of being at node x after two hops as $p'_x(u) = \sum_v p_v w_{u,v,x}$. To summarize, given the probability distribution over the predecessor node s, we compute the probability distribution over the next two hops to v and x. Finally, we can compute $\bar{p}_x = \sum_u \frac{1}{2}(p_x(u) + p'_x(u))$, the amount that node x appears overall in these two hops, summed over all u.

The final point is how we compute the predecessor distribution $q_{u,v}$. We don't want to have to do an expensive computation here to compute the stationary distribution every time we update the weights, so instead we update it iteratively along with the weights. In each iteration, we update q based on the current w and on the current estimate of q. Having done this, we then update the weights in a way that is analogous to the weighted walk, increasing weights to the nodes which receive too few visits (with respect to uniform visitation) and decreasing the weights to nodes that receive too many visits.

1: $q'_{v,u} \leftarrow \sum_s q_{u,s}w_{s,u,v}$
2: $q_{v,u} \leftarrow \frac{q'_{v,u}}{\sum_{u'} q'_{v,u'}}$
3: $p_v(u) \leftarrow \sum_s q_{u,s}w_{s,u,v}$.
4: $p'_x(u) \leftarrow \sum_v p_v w_{u,v,x}$
5: $\bar{p}_x \leftarrow \sum_u (p_x(u) + p'_x(u))/2$
6: $w'_{u,v} \leftarrow \frac{w_{u,v}}{\bar{p}_v}$
7: $w_{u,v} \leftarrow \frac{w'_{u,v}}{\sum_v w'_{u,v}}$

4. RESULTS

To evaluate these algorithms, we turn to a small subset of the entire Facebook social graph. All of the data used here was used anonymously. We would like to evaluate these methods on a portion of the graph that is manageable enough that rapid cycles of experimentation are feasible, but large enough that it captures the micro and macro structures of the full graph. One way to do this is to take the subgraph induced by the population of a single country, or a small set of countries. As one would expect, most of a typical individual's friends live in the same country as them, and the result of this is that taking the country subgraph does not remove a very large fraction of the edges incident to the nodes in that country.

In our experiments we pick four small but well-connected Central American countries: Guatemala (GT), Honduras (HN), El Salvador (SV) and Nicaragua (NI). For ease of experimentation, we use the state of the graph in July 2008, giving a smaller and more manageable dataset. As a final pre-processing step, we take the 10-core of the graph (iteratively remove nodes with degree less than 10). We define $d = 10$ as the threshold; if we did not take the 10-core there would be nodes that could not ever achieve this.

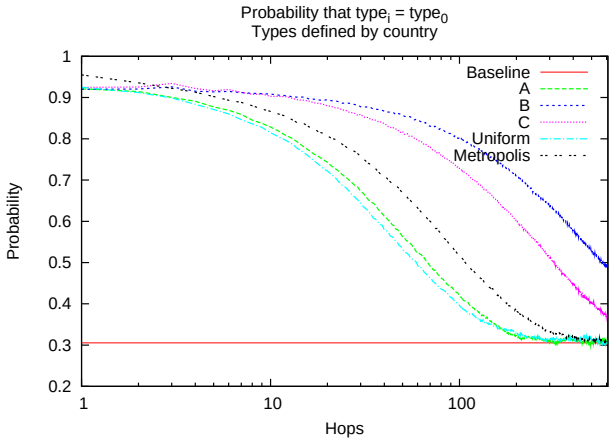

Figure 3: Probability of being in the start country.

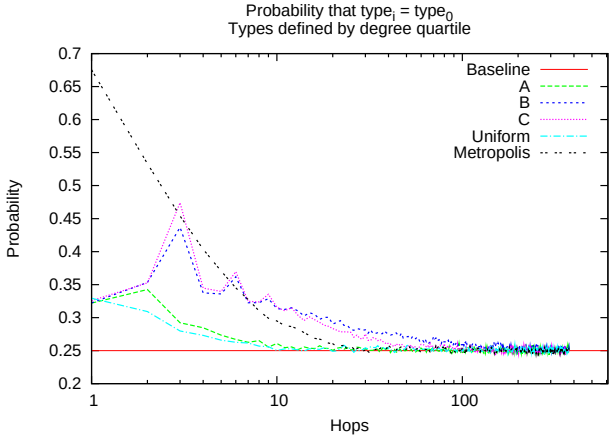

Figure 4: Probability of being in the same degree quartile.

The resulting graph has 117576 nodes, and 9406734 edges, over 90% of which have both end points in the same country.

Country	Nodes	Avg. Degree	Internal
GT	50102	84.2	93.1%
NI	11379	69.9	91.9%
HN	26075	80.8	92.3%
SV	30020	76.1	91.2%

In evaluating our algorithms, we first assign a type to each node according to one of two schemes. The first assignment scheme is based on the country the node is in. The second is based on which quartile the node's degree falls in. Each of these is a simplified, and stylized, simulation of the kind of variation one might find in a real test — with the variation based on geography in the first case, and based on the level of structural involvement in Facebook, as measured by degree, in the second case. (Clearly in real applications the sources of variation would be more complex, but these are both very natural parameters to use in our experiments.) Having assigned a type to each node, we next imagine that each type is associated with some coin bias, which defines the distributions X_u of the nodes u with this type. We evaluate two different sets of coin biases. First, we assign biases 4%, 5%, 6% and 7% to the four

different types. Second we assign biases 40%, 50%, 60% and 70%.

Assigning all of the nodes to a type, and hence a coin bias to each node, provides the test conditions for the algorithms. We then use the algorithms to sample nodes (both the core set and the supporting fringe) until our budget is exhausted. In our experiments we use a budget of 1000 users, or about 1% of all nodes. It is important to point out here that none of the sampling algorithms make explicit use of the type assignment; this is the unknown that has to be estimated, as it would need to be in a real test. They sample based only on properties of the network, while the type assignment independently assigns biases to the sampled nodes.

Having sampled a multiset of nodes, we randomly flip all of the coins according to their biases. We then estimate the overall population mean as $\sum \frac{X_u l_u}{p_u} / \sum \frac{l_u}{p_u}$. We perform this operation one million times for each algorithm (except for isolated sampling, where we can compute directly), and report the variance in our estimates over these one million trials. Because all of our estimates are unbiased, the mean of the estimates converges to the true population mean $E[X]$.

Variations on the testing procedure. We have also tried a variation on the model in which the nodes in the core set must have at least d neighbors in the fringe — that is, core-set nodes cannot contribute to the degree of other core-set nodes for purposes of meeting the degree threshold of d. Such a constraint is relevant to settings in which we plan to apply a new feature to members of the fringe, but withhold it from members of the core set, so as to separate the "senders" of the feature (the fringe nodes) from the measured "recipients" (the core nodes). We find that the relative performance of the different methods is essentially the same in this experimental treatment, with the main difference that the optimal lengths of walk segments is significantly shorter here. For the rest of the section, we discuss the standard experimental set-up, where we do count edges between nodes in the core toward the degree thresholds.

4.1 Walk Properties

Before giving performance results, we examine the properties of the walks themselves. We start by looking at how uniform our walk strategies are, as more uniform walks will tend to give lower variance estimates. We know *a priori* that the Metropolis walk is perfectly uniform and that the unweighted random walk is biased. We can quantify how biased the unweighted walk is and also how closely our algorithms approximate uniformity by computing the variance over the expected visitation distribution p. We look at walks of length 55, where we find that the unweighted walk has variance 2934. By contrast, Algorithm 1 has variance 0.0025. When we bias towards triangle closing in Algorithm 2, that variance increases to 634, but when we relearn the weights taking the triangle-closing bias into account, the variance of Algorithm 3 drops back to 0.015. Thus, our two methods for achieving uniformity in the walk do so nearly perfectly, with the remaining variance likely due to the fact that we don't completely converge in 100 iterations of our algorithms.

Next, we look at repetition in the walks. All other things being equal, we would achieve the best variance if the nodes all had multiplicity 1. Figure 1 shows the fraction of nodes that are unique in random walks for various lengths. We see here that the unweighted random walks, as well as the Algorithm 1 walks tend to mostly visit unique nodes, with over 90% of the hops being to new nodes, even after hundreds of hops. On the other hand, Metropolis sampling frequently stalls, and so even the first hop is relatively unlikely to visit a new node. In the middle, the triangle-closing algorithms tend to form more compact sets of nodes, and hence the revisit

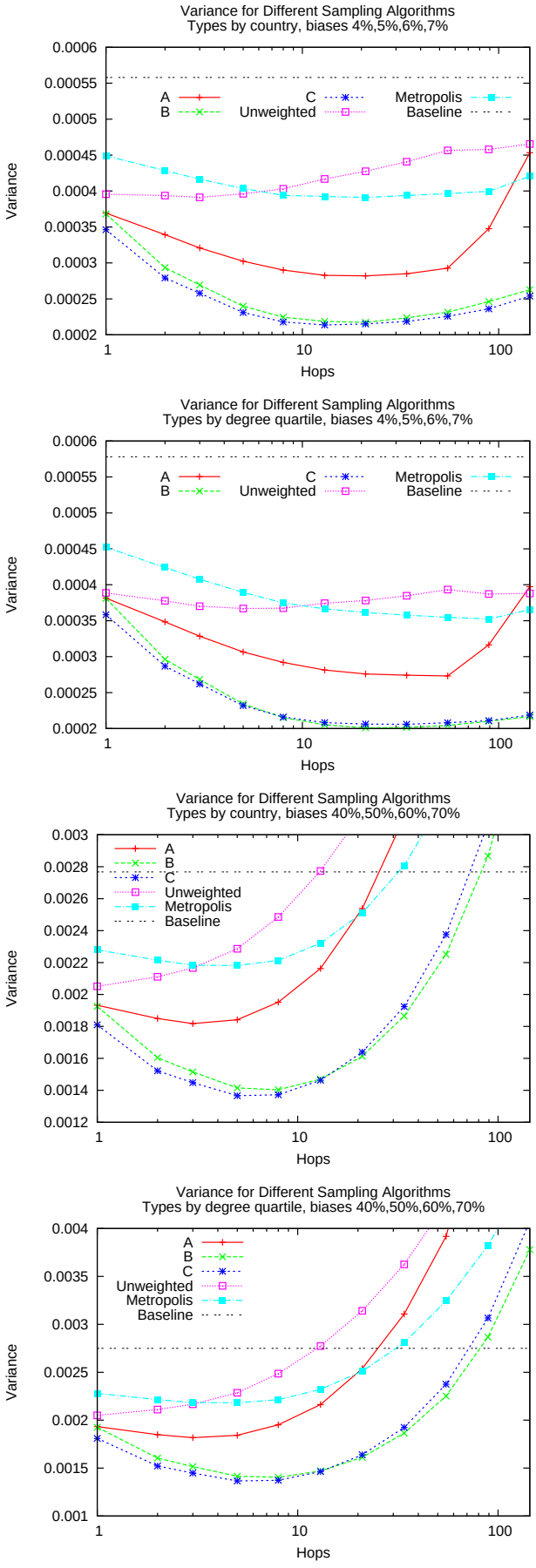

Figure 5: Variance under different conditions

nodes fairly frequently.

The other important property of the walks is their efficiency in terms of the number of supporting fringe nodes required per unique node visited. Figure 2 shows that all the walks become more efficient as they become longer. As expected, the triangle-closing walks are more efficient than their rivals.

4.2 Sample Variance

Types by Degree Quartile. In any system, some users are more active than others, and it is expected that power users will respond differently from more lightly engaged users. One simple proxy for engagement level is the degree of the nodes. More heavily engaged users will tend to have more connections in the social network.

Thus, in this section we examine the case where the response to the test is biased by degree. As high-degree nodes can easily have low-degree friends, one might expect that walks would be rapidly mixing with respect to node type, and hence the best solution would be to take a single long walk. Indeed, if each hop took you to a node with a degree selected uniformly from the four quartiles, longer walks would be better, owing to their increased efficiency in terms of budget spent per node evaluated. However, Figure 4 shows the probability of being at a node with the same type as the start node after h hops, as a function of h. While the walks all converge to 0.25 as expected, there is still some correlation between the start node and the 10^{th} node in the walk, suggesting that restarts will be helpful.

When we actually run the process and compute the variance, (see Figure 5) we find that intermediate length walks perform best, as they best trade off efficient use of budget with independence. In this experiment, the users in the bottom degree quartile are given bias 4% or 40% while the users in the top degree quartile are given bias 7% or 70%. For both sets of biases, we see that the triangle-closing walks perform significantly better than the alternatives and that Algorithm 1 outperforms all three baselines. For these two choices of biases, we find that the optimal number of hops is longer for the smaller set of biases. When we plug the variance expressions $E(V)$ and $V(E)$ from Section 2 into the formulae from that section, we find agreement between the model and this empirical observation; we discuss this further at the end of Section 4

One final note is that, because the walks come in discrete chunks, we may not be able to use all of our budget for long walks. This is particuarly apparent when looking at the long walks for Algorithm 1; if the core and fringe from one walk requires 501 nodes, and our budget is 1000, we can not take a second walk, and hence make end up with a significant unspent portion.

Types by Country. Here we perform our experiments identically, with the exception that the biases on the nodes are given by the country they reside in. The primary difference between this assignment of types and that by degree is that the country-based types are more correlated along the walk. Figure 3 shows that even after 100 hops, the country that a walk is in is highly correlated to the country that the walk starts in. In the case of the triangle-closing algorithms, the correlation continues out beyond 1000 hops. The consequence of this is that we expect the optimal number of hops to be shorter than it was in the case where we assigned types by degree, since we need to restart more often to ensure we get the right mix of types.

Figure 5 shows that the results match this intuition. The triangle-closing algorithms still perform best, with the more uniform version slightly outperforming the biased version. As expected, the optimal number of hops is smaller than it was in case where we assigned type by degree, and the variance increases sharply as the number of hops grows large.

4.3 Impact of budget

As we saw when we looked at the abstract bag-of-coins problem, the optimal solution is typically quite wasteful and this suggests that the best thing to do is to take many relatively short independent walks. The theory says that this should be the case regardless of budget. However, in principle it could turn out differently on real data for multiple reasons. First, the sets of nodes we sample do not all have the same type, and for long walks the distribution of types becomes well-mixed. Another difference is that the bags of coins did not have any multiplicities, whereas our model does, especially the triangle-closing versions.

We can test this empirically by finding the optimal hop length t^* for different budgets (we do this on Algorithm 2) and compare this value with the theoretical optimum $\widetilde{t}$ from the model in Section 2. To use the abstract model, we need a functional form for $g(t)$ from Section 2, and we find that $g(t) = 0.36t + 2.88t^{0.35}$ provides an excellent fit. Next, we compute $V(E)$ and $E(V)$ based on the type distribution (done by country in this experiment) and the biases (40% - 70%).

Plugging in the numbers, we find the optimal $\widetilde{t} = 9.4$. The table below shows the empirical optimum hop count t^* for different budgets, along with the empirical and theoretical variances. While the the variances are slightly worse (due to multiplicities in sampling), the optimal number of hops is empirically 8 (for budgets where discretization effects are not a significant factor), which typically yields 9 nodes, exactly matching the theoretical $\widetilde{t}$.

Budget	Optimal Hops	Empirical Var.	Model Var.
50	4	3.8E-2	2.4E-2
100	5	1.6E-2	1.2E-2
200	7	7.4E-3	5.9E-3
500	8	2.8E-3	2.4E-3
1000	8	1.4E-3	1.2E-3
2000	8	7.0E-4	5.9E-4
5000	7	2.8E-4	2.4E-4
10000	8	1.4E-4	1.2E-4

5. REFERENCES

[1] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. *Proc. ACM SIGKDD*, 2006.

[2] Z. Bar-Yossef, A. C. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about Web pages via random walks. *Proc. VLDB*, 2000.

[3] S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing Markov chain on a graph. *SIAM Review*, 46(4):667–689, 2004.

[4] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A case study of unbiased sampling of OSNs. *Proc. IEEE INFOCOM*, 2010.

[5] M. Henzinger, A. Heydon, M. Mitzenmacher, M. Najork. On near-uniform URL sampling. *Proc. WWW*, 2000.

[6] J. Leskovec, L. Adamic, and B. Huberman. The dynamics of viral marketing. *ACM Trans. Web*, 1(1), May 2007.

[7] N. Linial, A. Samorodnitsky, and A. Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. In *Proc. 30th ACM Symposium on Theory of Computing*, 1998.

[8] L. Lovász. Random walks on graphs: A survey. In *Combinatorics: Paul Erdös is Eighty*, 1996.

[9] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Annals of Mathematical Statistics*, 35(2):876–879, 1964.