CCIT Report #598
September 2006

Random Sampling from a Search Engine's Corpus*

Ziv Bar-Yossef† Maxim Gurevich‡

August 23, 2006

Abstract

We revisit a problem introduced by Bharat and Broder almost a decade ago: how to sample random pages from the corpus of documents indexed by a search engine, using only the search engine's public interface? Such a primitive is particularly useful in creating objective benchmarks for search engines.

The technique of Bharat and Broder suffers from a well-recorded bias: it favors long documents. In this paper we introduce two novel sampling algorithms: a lexicon-based algorithm and a random walk algorithm. Our algorithms produce *biased* samples, but each sample is accompanied by a *weight*, which represents its bias. The samples, in conjunction with the weights, are then used to *simulate* near-uniform samples. To this end, we resort to four well-known Monte Carlo simulation methods: *rejection sampling*, *importance sampling*, the *Metropolis-Hastings* algorithm, and the *Maximum Degree* method.

The limited access to search engines force our algorithms to use bias weights that are only "approximate". We characterize analytically the effect of approximate bias weights on Monte Carlo methods and conclude that our algorithms are *guaranteed* to produce near-uniform samples from the search engine's corpus. Our study of approximate Monte Carlo methods could be of independent interest.

Experiments on a corpus of 2.4 million documents substantiate our analytical findings and show that our algorithms do not have significant bias towards long documents. We use our algorithms to collect fresh comparative statistics about the corpora of the Google, MSN Search, and Yahoo! search engines.

1 Introduction

The latest round in the search engine size wars (cf. [36]) erupted in August 2005 after Yahoo! claimed [33] to index more than 20 billion documents. At the same time Google reported only 8 billion pages in its index, but simultaneously announced [5] that its index is three times larger than

*A preliminary version of this paper appeared in the proceedings of the 15th International World-Wide Web Conference (WWW2006) [4].

†Department of Electrical Engineering, Technion, Haifa 32000, Israel. Email: zivby@ee.technion.ac.il. Supported by the European Commission Marie Curie International Re-integration Grant.

‡Department of Electrical Engineering, Technion, Haifa 32000, Israel and IBM Research Lab in Haifa, Haifa 31905, Israel. Email: gmax@tx.technion.ac.il.

its competition's. This surreal debate underscores the lack of widely acceptable benchmarks for search engines.

Current evaluation methods for search engines [20, 23, 11] are labor-intensive and are based on anecdotal sets of queries or on fixed TREC data sets. Such methods do not provide statistical guarantees about their results. Furthermore, when the query test set is known in advance, search engines can manually adapt their results to guarantee success in the benchmark.

In an attempt to come up with reliable automatic benchmarks for search engines, Bharat and Broder [6] proposed the following problem: can we sample random documents from the corpus of documents indexed by a search engine, using only the engine's public interface? Unlike the manual methods, random sampling offers statistical guarantees about its test results. It is important that the sampling is done only via the public interface, and not by requesting the search engine itself to collect the sample documents, because we would like the tests to be objective and not to rely on the goodwill of search engines. Furthermore, search engines seem to be reluctant to allow random sampling from their corpus, because they do not want third parties to dig into their data.

Random sampling can be used to test the quality of search engines under a multitude of criteria: (1) *Overlap and relative sizes*: we can find out, e.g., what fraction of the documents indexed by Yahoo! are also indexed by Google and vice versa. Such size comparisons can indicate which search engines have better recall for narrow-topic queries. (2) *Topical bias*: we can identify themes or topics that are overrepresented or underrepresented in the corpus. (3) *Freshness evaluation*: we can evaluate the freshness of the corpus, by estimating the fraction of "dead links" it contains. (4) *Spam evaluation*: using a spam classifier, we can find the fraction of spam pages in the corpus. (5) *Security evaluation*: using an anti-virus software, we can estimate the fraction of indexed documents that are contaminated by viruses.

Random sampling can be used not only by third parties, but also by search service providers themselves to test the quality of their search engines and to compare them against the competition. The results of the comparison can then be used to improve the quality of the index. Although providers have full access to their own indices, using the public interface has several benefits: (1) It may be technically simpler to use this interface rather than to implement a procedure that directly samples from the corpus. (2) Applying the same sampling procedure to both the provider's own corpus and the competitors' corpora yields comparable results.

The Bharat-Broder approach Bharat and Broder proposed the following simple algorithm for uniformly sampling documents from a search engine's corpus. The algorithm successively formulates "random" queries, submits the queries to the search engine, and picks uniformly chosen documents from the result sets returned. In order to construct the random queries, the algorithm requires the availability of a *lexicon* of terms that appear in web documents. Each term in the lexicon should be accompanied by an estimate of its frequency on the web. Random queries are then formulated by randomly selecting a few terms from the lexicon, based on their estimated frequencies, and then taking their conjunction or disjunction. The lexicon is constructed in a pre-processing step by crawling a large corpus of documents (Bharat and Broder crawled the Yahoo! directory).

As Bharat and Broder noted in the original article [6] and was later confirmed by subsequent studies [12, 40], the method suffers from severe biases. The most significant bias is towards long, "content-

rich", documents, simply because these documents match many more queries than short documents. An extreme example is online dictionaries and word lists (such as the ispell dictionaries), which will be returned as the result of almost any query. Worse than that, when queries are formulated as conjunctions or disjunctions of unrelated terms, typically *only* dictionaries and word lists match the queries. Another problem is that search engines do not allow access to the full list of results, but rather only to the top k ones (where k is usually 1,000). Therefore, the Bharat-Broder algorithm may be biased towards documents with high static rank, if the random queries it generates tend to return more than k results. To alleviate this problem, Bharat and Broder used the estimated term frequencies to choose queries that are unlikely to return more than k results. Yet, collecting accurate term statistics is a major problem by itself, and thus if the estimated frequencies are off, then the bias towards highly ranked documents could be unavoidable.

Our contributions We propose two novel algorithms for sampling pages from a search engine's corpus. Both algorithms use a lexicon to formulate random queries, but unlike the Bharat-Broder approach, do not need to know term frequencies. The first algorithm, like the Bharat-Broder algorithm, requires a pre-processing step to build the lexicon. The second algorithm is based on a random walk on a virtual graph defined over the documents in the corpus. This algorithm does not need to build the lexicon a priori, but rather creates it "on the fly", as the random walk proceeds from document to document.

Both algorithms share the same basic framework. The algorithm first produces *biased* samples. That is, some documents are more likely to be sampled than others. Yet, each sample document x is accompanied by a corresponding "weight" $w(\mathrm{x})$, which represents the bias in the sample x. The weights allow us to apply *stochastic simulation* methods on the samples and consequently obtain *uniform*, unbiased, samples from the search engine's corpus.[1]

A simulation method accepts samples taken from a *trial distribution* p and simulates sampling from a *target distribution* π. In order to carry out the simulation, the simulator needs to be able to compute a "bias weight" $w(\mathrm{x}) = \pi(\mathrm{x})/p(\mathrm{x})$, for any given instance x. The simulator uses these weights to "promote" certain samples from p, while "demoting" other samples, thereby eliminating the initial bias in the trial distribution. The simulation has some overhead, which depends on how far p and π are from each other. In our case π is the uniform distribution over the search engine's corpus, while p is some other easy-to-sample-from distribution over the corpus. We employ four Monte Carlo simulation methods: *rejection sampling*, *importance sampling*, the *Metropolis-Hastings* algorithm, and the *Maximum Degree* method.

One technical difficulty in applying simulation methods in our setting is that the weights produced by our samplers are only *approximate*. To the best of our knowledge, stochastic simulation with approximate weights has not been addressed before. We are able to show that all four Monte Carlo methods still work even when provided with approximate weights. The distribution of the samples they generate is no longer identical to the target distribution π, but is rather only close to π. We provide extensive theoretical analysis of the effect of the approximate bias weights on the quality and the performance of the Monte Carlo methods. This study may be of independent interest.

[1] In fact, our algorithms are more general. They can be used to generate samples from any target distribution over the corpus, not only the uniform one.

Pool-based sampler A *query pool*, or a *lexicon*, is a collection of queries. Our pool-based sampler assumes knowledge of some query pool $\mathcal{P}$. The terms constituting queries in the pool can be collected by crawling a large corpus, like the Yahoo! [42] or ODP [15] directories. We stress again that knowledge of the frequencies of these terms is not needed.

The *degree* of a document x in the corpus is the number of queries from the pool that it matches. The "document degree distribution" is a distribution over the corpus, where each document is selected proportionally to its degree. The inner subroutine of the pool-based sampler generates samples from the document degree distribution. The corresponding bias weights are inverse-proportional to the document degrees, and are thus easy to calculate. The outer subroutine of the pool-based sampler then applies a Monte Carlo method (e.g., rejection sampling) on the samples from the document degree distribution in order to simulate uniform samples.

How does the algorithm generate samples from the document degree distribution? The first solution that comes to mind is as follows. The algorithm picks a random query from the pool, submits the query to the search engine, and then selects a random document from the result set returned. Indeed, documents with high degree match more queries, and are thus more likely to be sampled than documents with low degree. However, the resulting sampling distribution is not exactly the document degree distribution, as the chance of a document to be selected depends also on the number of results returned on queries that this document matches. For example, if documents x and x' have both degree 1, but x matches a query q with 100 results, while x' matches a query q' with 50 results, then the probability of x' to be sampled is twice as high as the probability of x to be sampled.

To alleviate this problem, the algorithm does not select the query uniformly at random, but rather proportionally to its *cardinality*. The cardinality of a query q is the number of results it has. It can be shown that if the algorithm samples queries from the pool proportionally to their cardinalities, then by selecting random documents from their result sets, the algorithm could have obtained samples from the document degree distribution. Sampling queries according to their cardinality is tricky, though, because we do not know a priori the cardinality of queries. What we do instead is sample queries from the pool uniformly, and then *simulate* sampling from the cardinality distribution. To this end, we use Monte Carlo methods again. Hence, Monte Carlo methods are used twice: first to generate the random queries and then to generate the uniform documents.

To demonstrate how the pool-based sampler works, consider a corpus consisting of only 100 documents and a query pool with two queries q_1 and q_2, whose cardinalities are 99 and 2, respectively. Suppose the result sets of $\mathrm{q}_1, \mathrm{q}_2$ share a single document x. The sampler chooses one of $\mathrm{q}_1, \mathrm{q}_2$ uniformly at random and then applies an acceptance-rejection procedure, in which q_1 is accepted with probability 99/100 and q_2 is accepted with probability 2/100. If the query is accepted, it is submitted to the search engine, and a random document is chosen from its result set. A second acceptance-rejection procedure is applied on this document. If it is the document x, it is accepted with probability 1/2, and otherwise it is accepted with probability 1. If the document is accepted, it is output as a sample. It can be verified that all 100 documents in the corpus are equally likely to be sampled.

We rigorously analyze the pool-based sampler and identify the important properties of the query pool that make this technique accurate and efficient. We find that using a pool of exact phrase queries of a certain length is much more preferable to using conjunctive or disjunctive queries, like

the ones used by Bharat and Broder.

Random walk sampler We present another sampler, which also uses a query pool, but does not need to construct it a priori. This sampler performs a random walk on a virtual graph defined over the documents in the corpus. The limit distribution of this random walk is the uniform distribution over the documents, and thus if we run the random walk for sufficiently many steps, we are guaranteed to obtain near-uniform samples from the corpus.

The graph is defined as follows: two documents are connected by an edge if and only if they match the same query from the pool. This means that if one submits the shared query to the search engine, both documents are guaranteed to belong to the result set. Running a random walk on this graph is simple: we start from an arbitrary document, at each step choose a random query that the current document matches, submit this query to the search engine, and move to a randomly chosen document from the query's result set.

The random walk as defined does not converge to the uniform distribution. In order to make it uniform, we apply either the Metropolis-Hastings algorithm or the Maximum Degree method. We provide careful analysis of the random walk sampler. Like the pool-based sampler, this sampler too is guaranteed to produce near-uniform samples. However, theoretical analysis of its performance suggests that it is less efficient than the pool-based sampler.

Experimental results To validate our techniques, we crawled 2.4 million English pages from the ODP hierarchy [15], and built a search engine over these pages. We used a subset of these pages to create the query pool needed for our pool-based sampler and for the Bharat-Broder sampler.

We ran our two samplers as well as the Bharat-Broder sampler on this search engine, and calculated bias towards long documents and towards highly ranked documents. As expected, the Bharat-Broder sampler was found to have significant bias. On the other hand, our pool-based sampler had no bias at all, while the random walk sampler only had a small negative bias towards short documents.

We then ran our pool-based sampler on Google [19], MSN Search [35], and Yahoo! [42]. As a query pool, we used 5-term phrases extracted from English pages at the ODP hierarchy. The samples collected enabled us to produce up-to-date estimates of the relative sizes of these search engines as well as interesting statistics about their freshness, their domain name distribution, and their coverage of dynamic URLs.

The rest of the paper is organized as follows. In Section 2 we review some related work. Section 3 overviews some tools from probability theory and statistics used in our analysis. In Section 4 we briefly review the four Monte Carlo simulation methods we use. In Section 5 we analyze the effect of approximate bias weights on Monte Carlo simulation. In Section 6 we describe a formal framework for studying search engine samplers. In Sections 7 and 8 we outline in detail our two samplers. Section 9 includes our experimental results, and Section 10 some concluding remarks.

In order to avoid disturbing the flow of the paper, most proofs are postponed to the appendix.

2 Related work

Apart from Bharat and Broder, several other studies used queries to search engines to collect random samples from their corpora. Queries were either manually crafted [9], collected from user query logs [28], or selected randomly using the technique of Bharat and Broder [21, 12]. Assuming search engine corpora are independent and uniformly chosen subsets of the web, estimates of the sizes of search engines and of the indexable web have been derived. Due to the bias in the samples, though, these estimates lack any statistical guarantees. Dobra and Fienberg [16] showed how to avoid the unrealistic independence and uniformity assumptions, but did not address the sampling bias. We believe that their methods could be combined with ours to obtain accurate size estimates.

Several studies [29, 24, 25, 3, 37] developed methods for sampling pages from the indexable web. Such methods can be used to also sample pages from a search engine's corpus. Yet, since these methods try to solve a harder problem, they also suffer from various biases, which our method does not have. It is interesting to note that the random walk approaches of Henzinger *et al.* [25] and Bar-Yossef *et al.* [3] implicitly use importance sampling and the Maximum Degree method, respectively, to make their samples near-uniform. Yet, the bias they suffer towards pages with high in-degree is significant.

Anagnostopoulos, Broder, and Carmel [2] proposed an enhancement to index architecture that could support random sampling from the result sets of broad queries. This is very different from what we do in this paper: our techniques do not propose any changes to current search engine architecture and do not rely on internal data of the search engine; moreover, our goal is to sample from the whole corpus and not from the result set of a particular query.

A recent study by Broder *el al.* [10] presents an algorithm for accurately estimating the absolute size of a search engine's corpus, using the engine's public interface. The cleverly crafted estimator uses our techniques to generate uniform samples from the search engine's corpus.

3 Preliminaries

In this section we outline our notations and conventions and review some tools from statistics that will be handy in our analysis.

3.1 Conventions and notations

Sets are denoted by uppercase calligraphic letters: $\mathcal{D}, \mathcal{Q}, \mathcal{P}$. Elements in sets are denoted by lowercase Roman letters: $\mathrm{x}, \mathrm{y}, \mathrm{q}$. Random variables are denoted by uppercase Roman letters: $\mathrm{X}, \mathrm{Y}, \mathrm{Q}$. Probability distributions are denoted by small italicized letters, Roman or Greek: p, q, π.

All probability spaces in this paper are discrete and finite. A *probability distribution* p on a finite probability space $\mathcal{U}$ is a function $p : \mathcal{U} \to [0, 1]$ s.t. $\sum_{\mathrm{x} \in \mathcal{U}} p(\mathrm{x}) = 1$. The *support of* p is defined as:

$$\mathrm{supp}(p) = \{\mathrm{x} \in \mathcal{U} \mid p(\mathrm{x}) > 0\}.$$

A subset $\mathcal{E}$ of the probability space $\mathcal{U}$ is called an *event*. For an event $\mathcal{E} \subseteq \mathcal{U}$, we define $p(\mathcal{E})$ to be the probability of this event under p:

$$p(\mathcal{E}) = \sum_{\mathrm{x} \in \mathcal{E}} p(\mathrm{x}).$$

A *random variable* with distribution p and range $\mathcal{V}$ is a function $\mathrm{X} : \mathcal{U} \to \mathcal{V}$. Unless stated otherwise, when we refer to a random variable with distribution p, we mean the identity random variable: $\mathcal{V} = \mathcal{U}$ and $\mathrm{X}(\mathrm{x}) = \mathrm{x}$, for all $\mathrm{x} \in \mathcal{U}$.

Given a predicate $f : \mathcal{V} \to \{0, 1\}$, $f(\mathrm{X})$ is the following event:

$$f(\mathrm{X}) = \{\mathrm{x} \in \mathcal{U} \mid f(\mathrm{X}(\mathrm{x})) = 1\}.$$

The probability of the event $f(\mathrm{X})$ under p is denoted $\Pr_p(f(\mathrm{X}))$.

When the range of the random variable is $\mathcal{V} = \mathbb{R}$, we can define the *expectation* of X under p:

$$\mathbb{E}_p(\mathrm{X}) = \sum_{\mathrm{x} \in \mathcal{U}} p(\mathrm{x})\ \mathrm{X}(\mathrm{x}).$$

The *variance* of X is defined as:

$$\mathrm{var}_p(\mathrm{X}) = \mathbb{E}_p((\mathrm{X} - \mathbb{E}_p(\mathrm{X}))^2) = \mathbb{E}_p(\mathrm{X}^2) - (\mathbb{E}_p(\mathrm{X}))^2.$$

The *standard deviation* of X is the square root of its variance:

$$\sigma_p(\mathrm{X}) = \sqrt{\mathrm{var}_p(\mathrm{X})}.$$

We define the *mean deviation* of a random variable X to be its expected absolute deviation from its mean:

$$\mathrm{dev}_p(\mathrm{X}) = \mathbb{E}_p(|\mathrm{X} - \mathbb{E}_p(\mathrm{x})|).$$

It follows from the Cauchy-Schwartz inequality that the mean deviation is always bounded by the standard deviation:

Proposition 1. *For any random variable X,* $\mathrm{dev}_p(X) \leq \sigma_p(X)$.

The *normalized mean deviation* of X is the mean deviation, normalized by the mean:

$$\mathrm{ndev}_p(\mathrm{X}) = \frac{\mathrm{dev}_p(\mathrm{X})}{\mathbb{E}_p(\mathrm{X})}.$$

The *covariance* of two random variables (X, Y) with joint distribution p is defined as:

$$\mathrm{cov}_p(\mathrm{X}, \mathrm{Y}) = \mathbb{E}_p(\mathrm{XY}) - \mathbb{E}_p(\mathrm{X})\,\mathbb{E}_p(\mathrm{Y}).$$

Throughout, we assume knowledge of basic probability theory. Everything we use can be found in any standard textbook on the subject.

3.2 Total variation distance

The *total variation distance* between two distribution p, q on the same space $\mathcal{U}$ is defined as:

$$||p - q|| = \frac{1}{2} \sum_{\mathrm{x} \in \mathcal{U}} |p(\mathrm{x}) - q(\mathrm{x})|.$$

We use total variation distance, because it has some nice properties described below. Yet, other statistical distance measures, like the Kullback-Leibler divergence could have been used as well.

The following is a standard characterization of the total variation distance:

Lemma 2. *Let p, q be two distributions on the same probability space $\mathcal{U}$. Then,*

$$||p - q|| = \max_{\mathcal{E} \subseteq \mathcal{U}} |p(\mathcal{E}) - q(\mathcal{E})|.$$

3.3 Wald's identity

Suppose we invoke a sampler T times, where T is a random variable, and each invocation requires n search engine queries in expectation. What is then the expected total number of search engine queries made? As T is a random variable, simple linearity of expectation cannot be used in the analysis. The following identity from statistical sequential analysis shows that the expectation equals $n \cdot \mathbb{E}(T)$.

Theorem 3 (Wald's identity). *Let $X_1, X_2, \ldots$ be an infinite sequence of independent and identically distributed random variables with mean μ. Let T be a random variable on $\{0, 1, 2, \ldots\}$, for which the event $\{T = k\}$ is independent of $X_{k+1}, X_{k+2}, \ldots$ for all k (T is called a* stopping time random variable*). We further assume $\mathbb{E}(T) < \infty$. Then,*

$$\mathbb{E}(\sum_{i=1}^{T} X_i) = \mu \, \mathbb{E}(T).$$

A proof of Wald's identity can be found, e.g., in the textbook of Siegmund [38], Section 2.2.

4 Monte Carlo methods

We briefly review the four Monte Carlo simulation methods we use in this paper. For a more elaborate overview, refer to the textbook of Liu [31].

4.1 Basic framework

The basic question addressed in stochastic simulation is the following. There is a *target distribution* π on a space $\mathcal{U}$, which is hard to sample from directly. On the other hand, there is an easy-to-sample-from *trial distribution* p on the same space $\mathcal{U}$. Can we then somehow use the samples from

p to *simulate* sampling from π? A Monte Carlo simulator is a procedure, which given samples from p generates samples from π. In order to carry out the simulation, the simulator requires access to three "oracle procedures", which we describe next.

The first procedure, $\texttt{getSample}_p()$, generates samples from the trial distribution p. Each invocation returns a single sample X from p. Successive invocations generate independent and identically distributed samples $\mathrm{X}_1, \mathrm{X}_2, \ldots$.

The two other oracle procedures are used to calculate *unnormalized forms* of the distributions π and p:

Definition 4 (Unnormalized form of a distribution). Let π be a distribution on a space $\mathcal{U}$. An *unnormalized form* of π is a function $\hat{\pi} : \mathcal{U} \to [0, \infty)$, which equals π up to a *normalization constant* $Z_{\hat{\pi}} > 0$. That is,

$$\forall \mathrm{x} \in \mathcal{U}, \quad \hat{\pi}(\mathrm{x}) = \pi(\mathrm{x}) \cdot Z_{\hat{\pi}}.$$

$\hat{\pi}(\mathrm{x})$ is a relative weight, which represents the probability of x to be chosen in the distribution π. For example, if π is the uniform distribution on $\mathcal{U}$, then all elements are equally likely to be selected. Hence, the straightforward unnormalized form of π is: $\hat{\pi}(\mathrm{x}) = 1$, for all $\mathrm{x} \in \mathcal{U}$. The corresponding normalization constant is $Z_{\hat{\pi}} = |\mathcal{U}|$.

A Monte Carlo simulator needs two oracle functions, $\texttt{getWeight}_{\hat{\pi}}(\mathrm{x})$ and $\texttt{getWeight}_{\hat{p}}(\mathrm{x})$, which given an instance $\mathrm{x} \in \mathcal{U}$ return the unnormalized weights $\hat{\pi}(\mathrm{x})$ and $\hat{p}(\mathrm{x})$, respectively. $\hat{\pi}$ and $\hat{p}$ are any unnormalized forms of π and p. Note that the simulator does *not* need to know the corresponding normalization constants $Z_{\hat{\pi}}$ and $Z_{\hat{p}}$.

4.2 Rejection sampling

Rejection sampling, due to John von Neumann [41], is the most classical Monte Carlo method. Rejection sampling makes two assumptions: (1) $\mathrm{supp}(\pi) \subseteq \mathrm{supp}(p)$; and (2) there is a known *envelope constant* C satisfying:

$$C \geq \max_{\mathrm{x} \in \mathrm{supp}(p)} \frac{\hat{\pi}(\mathrm{x})}{\hat{p}(\mathrm{x})}.$$

The procedure, described in Figure 1, repeatedly generates samples from the trial distribution p, until a sample is "accepted". To decide whether a sample X is accepted, the procedure applies an acceptance-rejection procedure. The procedure accepts the sample X with the following acceptance probability:

$$r_{\mathrm{RS}}(\mathrm{X}) = \frac{\hat{\pi}(\mathrm{X})}{C\ \hat{p}(\mathrm{X})}.$$

We call r_{RS} the *acceptance function*. Note that $r_{\mathrm{RS}}(\mathrm{x}) \in [0, 1]$, for all $\mathrm{x} \in \mathrm{supp}(p)$, due to the property of the envelope constant C.

Intuitively, rejection sampling uses the acceptance-rejection procedure to bridge the gap between p and π. For example, when π is the uniform distribution and p is some non-uniform distribution, then the procedure assigns high acceptance probabilities to instances that have low probability in p and low acceptance probabilities to instances that have high probabilities in p. Thus, the

```
1:Function RejectionSampling(C)
2:   while (true) do
3:     X := getSample_p()
4:     if (accept(C,X))
5:       return X

1:Function accept(C,x)
2:   r_RS(x) := π̂(x) / (C p̂(x))
3:   toss a coin whose heads probability is r_RS(x)
4:   return true if and only if coin comes up heads
```

Figure 1: The rejection sampling procedure.

acceptance-rejection procedure smoothes the distribution p. A simple analysis shows that for any π and p, the distribution of the accepted samples is *exactly* the target distribution π.

The expected number of samples from p needed in order to generate each sample of π is $CZ_{\hat{p}}/Z_{\hat{\pi}} \geq \max_{\mathrm{x}\in\mathcal{U}} \pi(\mathrm{x})/p(\mathrm{x})$. Hence, the efficiency of the procedure depends on two factors: (1) the similarity between the target distribution and the trial distribution: the more similar they are the smaller is $\max_{\mathrm{x}\in\mathcal{U}} \pi(\mathrm{x})/p(\mathrm{x})$; and (2) the gap between the envelope constant C and $\max_{\mathrm{x}\in\mathcal{U}} \hat{\pi}(\mathrm{x})/\hat{p}(\mathrm{x})$.

The main drawback of rejection sampling is the need to know the envelope constant C. A too high value makes the procedure inefficient, while a too low value violates the envelope condition.

4.3 Importance sampling

Importance sampling [32] does not generate samples from the target distribution π, but rather uses samples from p to directly estimate statistical parameters relative to the distribution π. For simplicity, we assume the desired statistical parameter is $\mathbb{E}_\pi(f(\mathrm{X}))$, where f is some real-valued function, although the technique is applicable to other statistical parameters as well. Unlike rejection sampling, there is no need to know an "envelope constant".

In order to estimate $\mathbb{E}_\pi(f(\mathrm{X}))$, the importance sampling procedure (see Figure 2) generates n independent samples $\mathrm{X}_1, \ldots, \mathrm{X}_n$ from the trial distribution p. If $p = \pi$, then clearly $\frac{1}{n}\sum_{i=1}^n f(\mathrm{X}_i)$ is an unbiased estimator of $\mathbb{E}_\pi(f(\mathrm{X}))$. However, when $p \neq \pi$, the samples $\mathrm{X}_1, \ldots, \mathrm{X}_n$ are "weighted" and the weights have to be accounted for in the estimation. The *importance ratio* at x, which is defined as

$$w(\mathrm{x}) = \frac{\hat{\pi}(\mathrm{x})}{\hat{p}(\mathrm{x})},$$

is exactly the desired weight. Hence, $\frac{1}{n}\sum_{i=1}^n f(\mathrm{X}_i)w(\mathrm{X}_i)$ is an unbiased estimator of $\mathbb{E}_\pi(f(\mathrm{X}))$, modulo normalization. In order to get a correct estimator, we need to estimate also the ratio between the normalization constants of $\hat{\pi}$ and $\hat{p}$. Hence, the final estimator is:

$$\hat{\mu} = \frac{\frac{1}{n}\sum_{i=1}^n f(\mathrm{X}_i)w(\mathrm{X}_i)}{\frac{1}{n}\sum_{i=1}^n w(\mathrm{X}_i)}.$$

1:**Function** ImportanceSampling(f,n)
2: for $i = 1$ to n do
3: $\mathrm{X}_i := \texttt{getSample}_p()$
4: $w(\mathrm{X}_i) := \frac{\hat{\pi}(\mathrm{X}_i)}{\hat{p}(\mathrm{X}_i)}$
5: compute $f(\mathrm{X}_i)$

6: output $\frac{\frac{1}{n}\sum_{i=1}^{n} f(\mathrm{X}_i)w(\mathrm{X}_i)}{\frac{1}{n}\sum_{i=1}^{n} w(\mathrm{X}_i)}$

Figure 2: The importance sampling procedure.

Remark. This is one of several possible importance sampling estimators. The estimator is biased (i.e., its expectation is not necessarily $\mathbb{E}_\pi(f(\mathrm{X}))$), however it is guaranteed to be close to the true value with high probability, as long as n is sufficiently large. See more details in [26].

The efficiency of importance sampling depends on how close is the "shape" of $\hat{p}(\mathrm{x})$ to the "shape" of $f(\mathrm{x})\hat{\pi}(\mathrm{x})$. An appropriately chosen p can lead to less samples than even sampling from π directly. See more details in [30]. In this paper we rely on the Liu's "rule of thumb" [30]. It states that the number of samples from p, required to estimate $\mathbb{E}_\pi(f(\mathrm{X}))$ with the same confidence level (variance) as if using n independent samples from π, is at most $n(1 + \mathrm{var}_p(\pi(\mathrm{X})/p(\mathrm{X})))$.

4.4 Markov Chain Monte Carlo methods

In some situations even generating i.i.d. samples from the trial distribution p is infeasible. Instead, we are given a *random walk* that converges in the limit to p. Can we then transform this random walk into a new random walk that converges to the target distribution π? This is the question addressed by Markov Chain Monte Carlo (MCMC) methods. In this paper we focus on two of these methods: the Metropolis-Hastings (MH) algorithm [34, 22] and the Maximum Degree (MD) method (cf. [3, 8]).

A *Markov Chain* (a.k.a. *random walk*) on a finite state space $\mathcal{U}$ is a stochastic process in which states of $\mathcal{U}$ are visited successively. The Markov chain is specified by a $|\mathcal{U}| \times |\mathcal{U}|$ *probability transition matrix* P. P is a *stochastic matrix*, meaning that every row x of P specifies a probability distribution P_x on $\mathcal{U}$. P induces a directed graph G_P on $\mathcal{U}$ with non-negative edge weights. There is an edge $\mathrm{x} \to \mathrm{y}$ in the graph if $P(\mathrm{x}, \mathrm{y}) > 0$ and the corresponding weight is $P(\mathrm{x}, \mathrm{y})$.

The Markov chain is called *ergodic*, if it satisfies two conditions: (1) it is *irreducible*, meaning that the graph G_P is strongly connected; and (2) it is *aperiodic*, meaning that the g.c.d. of the lengths of directed paths connecting any two nodes in G_P is 1.

A random walk process starts at some initial state $\mathrm{x}_0 \in \mathcal{U}$. The initial state is chosen according to an initial state distribution p_0 on $\mathcal{U}$ (typically, the mass of the initial distribution is concentrated on a single fixed state of $\mathcal{U}$). The random walk then successively moves between states of $\mathcal{U}$. After visiting state x, the next state is chosen randomly according to the probability distribution P_x. This process goes on indefinitely.

Each step t of the random walk induces a probability distribution p_t on the state space $\mathcal{U}$. The

initial distribution is p_0. Successive distributions are given by the recursive formula: $p_t = p_{t-1}P$. Therefore, $p_t = p_0 P^t$. A fundamental theorem of the theory of Markov chains states that if a Markov chain is ergodic, then *regardless* of the initial distribution p_0, the sequence of distributions $p_0, p_1, p_2, \ldots$ is guaranteed to converge to a unique limit distribution p. That is,

$$||p_t - p|| \stackrel{t\to\infty}{\longrightarrow} 0.$$

Furthermore, the unique limit distribution p is also the unique *stationary distribution* of P:

$$pP = p.$$

A random walk sampler (see Figure 3) uses a Markov chain to generate samples from a distribution which is close to p. The algorithm starts the random walk from any given initial state x_0 and runs it for B steps (B is called the "burn-in period"). The reached state x_B is then returned as the sample. By the above, the distribution of this sample is p_B, and thus if B is sufficiently large, $||p_B - p||$ is small. (We discuss below how to choose a sufficiently large burn-in period.) To generate more samples, the algorithm runs the random walk again and again. (There are more efficient sampling procedures, which we mention below.)

1:**Function** RandomWalk(P, B, x_0)
2: X := x_0
3: for $t = 1$ to B do
4: Y := sample generated according to P_{X}
5: X := Y
6: return X

Figure 3: A random walk sampler.

MCMC methods allow us to transform a given ergodic Markov chain P whose limit distribution is p into a new Markov chain P_{MCMC} whose limit distribution is π. The two MCMC methods we consider in this paper, MH and MD, use the same framework to perform the transformation. The MCMC sampler (see Figure 4) runs a random walk similarly to the random walk sampler, except that it applies an acceptance-rejection procedure at each step. The procedure is used to determine whether the "proposed state" Y is "accepted" and thus will become the next step of the random walk or not. The acceptance function $r_{\mathrm{MCMC}}(\mathrm{x}, \mathrm{y})$ depends on both the current state and the proposed state. MH and MD differ in the choice of the acceptance function.

The acceptance-rejection procedure effectively modifies the transition probabilities of the random walk and defines a new transition matrix P_{MCMC}. A careful choice of the acceptance function r_{MCMC} guarantees that the limit distribution of P_{MCMC} is the target distribution π.

4.4.1 The Metropolis-Hastings algorithm

The MH algorithm requires that the initial Markov chain P is not only ergodic but also satisfies the following condition: for all states $\mathrm{x}, \mathrm{y} \in \mathcal{U}$, $P(\mathrm{x}, \mathrm{y}) > 0 \Leftrightarrow P(\mathrm{y}, \mathrm{x}) > 0$. Also the supports of the limit distribution p and of the target distribution π must equal the whole state space (i.e., $\mathrm{supp}(p) = \mathrm{supp}(\pi) = \mathcal{U}$).

1:**Function** MCMC(P, B, x_0)
2: X := x_0
3: for $t = 1$ to B do
4: Y := sample generated according to P_{X}
5: if (accept(P,X,Y))
6: X := Y
7: return X

1:**Function** accept(P, x, y)
2: compute $r_{\mathrm{MCMC}}(\mathrm{x},\mathrm{y})$ from $\hat{\pi}(\mathrm{x})$, $\hat{\pi}(\mathrm{y})$, $\hat{p}(\mathrm{x})$, $\hat{p}(\mathrm{y})$, $P(\mathrm{x},\mathrm{y})$, and $P(\mathrm{y},\mathrm{x})$.
3: toss a coin whose heads probability is $r_{\mathrm{MCMC}}(\mathrm{x},\mathrm{y})$
4: return true if and only if coin comes up heads

Figure 4: An MCMC sampler.

The acceptance function used by the MH algorithm is the following:

$$r_{\mathrm{MH}}(\mathrm{x},\mathrm{y}) = \min\left\{\frac{\pi(\mathrm{y})\ P(\mathrm{y},\mathrm{x})}{\pi(\mathrm{x})\ P(\mathrm{x},\mathrm{y})},\ 1\right\}.$$

Note that since $\frac{\hat{\pi}(\mathrm{y})}{\hat{\pi}(\mathrm{x})} = \frac{\pi(\mathrm{y})}{\pi(\mathrm{x})}$, oracle access to an unnormalized form of π suffices for computing this acceptance function.

The probability transition matrix of the resulting Markov chain is:

$$P_{\mathrm{MH}}(\mathrm{x},\mathrm{y}) \ = \ \begin{cases} P(\mathrm{x},\mathrm{y})\ r_{\mathrm{MH}}(\mathrm{x},\mathrm{y}), & \text{if } \mathrm{x} \neq \mathrm{y}, \\ P(\mathrm{x},\mathrm{x})\ r_{\mathrm{MH}}(\mathrm{x},\mathrm{x}) + 1 - \sum_{\mathrm{z}\in\mathcal{U}} P(\mathrm{x},\mathrm{z})\ r_{\mathrm{MH}}(\mathrm{x},\mathrm{z}), & \text{if } \mathrm{x} = \mathrm{y}. \end{cases}$$

It can be shown (see, e.g., [14]) that the limit distribution of this Markov chain is exactly π.

4.4.2 The Maximum Degree method

The MD method applies to arbitrary ergodic Markov chains. The supports of the limit distribution p and of the target distribution π should satisfy $\mathrm{supp}(p) = \mathrm{supp}(\pi) = \mathcal{U}$. Similarly to rejection sampling, application of this method requires the availability of an "envelope constant" C. C must satisfy the following condition:

$$C \ \geq \ \max_{\mathrm{x}\in\mathcal{U}} \frac{\hat{p}(\mathrm{x})}{\hat{\pi}(\mathrm{x})}.$$

The need for an envelope constant is the major disadvantage of the Maximum Degree method relative to the Metropolis-Hastings algorithm. However, if the envelope constant is chosen sufficiently close to its lower bound, then the MD method can become significantly more efficient than the MH algorithm.

The acceptance function used by the MD algorithm is the following:

$$r_{\mathrm{MD}}(\mathrm{x}) = \frac{\hat{p}(\mathrm{x})}{C\ \hat{\pi}(\mathrm{x})}.$$

Remark. Notice the reversed roles of $\hat{p}(\mathrm{x})$ and $\hat{\pi}(\mathrm{x})$, comparing to the rejection sampling acceptance function. This is not accidental. Rejection sampling is similar (but not identical) to the application of the MD method on a degenerate random walk, which converges in one step to the limit distribution p (that is, all the rows of the transition matrix P equal p). The reversed roles are due to the reversed semantics of acceptance in rejection sampling versus MCMC methods. In rejection sampling, when the current state is accepted, the process stops and outputs the current state. In MCMC methods, when a *proposed* state is accepted, then implicitly the current state is *rejected*, and the process continues.

The probability transition matrix of the MD Markov chain is:

$$P_{\mathrm{MD}}(\mathrm{x},\mathrm{y}) \;=\; \begin{cases} P(\mathrm{x},\mathrm{y})\; r_{\mathrm{MD}}(\mathrm{x}), & \text{if } \mathrm{x} \neq \mathrm{y}, \\ P(\mathrm{x},\mathrm{x})\; r_{\mathrm{MD}}(\mathrm{x}) + 1 - r_{\mathrm{MD}}(\mathrm{x}), & \text{if } \mathrm{x} = \mathrm{y}. \end{cases}$$

Again, it can proved that the limit distribution of this Markov chain is π.

Remark. To the best of our knowledge, the formulation above is the first application of the Maximum Degree method to arbitrary ergodic Markov chains and to arbitrary target distributions. Previous studies [3, 8] applied the MD method only in the special case P is a simple random walk on an undirected graph G and π is the uniform distribution over the vertices of G. The limit distribution of the simple random walk is the degree distribution (i.e., each node is chosen proportionally to its degree). In this case C must be set as an upper bound on the maximum degree of the graph, and that is why the method is called "Maximum Degree".

The acceptance function of the MD method has the peculiar property that it depends only on the current state x and not on the proposed state y. This fact allows a more efficient implementation of the MD sampler (see Figure 5). This sampler postpones the selection of the proposed state Y to until after acceptance is achieved. Hence, rather than selecting a proposed state every time the acceptance-rejection procedure is called, the proposed state is selected only once. Since in many situations selection of a proposed state is costly, this amounts to significant savings in running time.

A further improvement is possible. Note that the number of steps the random walk spends at each state x is a geometric random variable with a known success probability. Therefore, when the sampler moves to a new state x, it can *simulate* the number of steps that it is going spend at the state by generating an appropriate geometric random variable. It can then immediately select the next state. This saves the need to perform the iterative coin tosses.

4.4.3 The burn-in period

How should we set the burn-in period B of a random walk in order to guarantee that the selected sample has distribution which is close to the limit distribution? To this end, a rich pool of techniques, ranging from algebraic to geometric, is available from the theory of Markov chains (see a survey by Diaconis and Saloff-Coste [14] for a detailed review). In this paper we focus on a popular technique, which is based on estimating the *spectral gap* of the Markov chain's transition matrix.

Let P be the transition matrix of an ergodic Markov chain, whose limit distribution is p. For each $\varepsilon > 0$, we define the *ε-burn-in period* (a.k.a. *ε-mixing time*) of the Markov chain as:

$$T_\varepsilon(P) = \min\{t \mid \text{for all initial distributions } p_0 \text{ and } \forall t' \geq t,\ ||p_{t'} - p|| < \varepsilon\}.$$

```
1:Function MD(P, B, x_0, C)
2:  X := x_0
3:  for t = 1 to B do
4:    if (accept(P,C,X))
5:      Y := sample generated according to P_X
6:      X := Y
7:  return X

1:Function accept(P, C, x)
2:   r_MD(x) := p̂(x) / (C π̃(x))
3:   toss a coin whose heads probability is r_MD(x)
4:   return true if and only if coin comes up heads
```

Figure 5: The Maximum Degree sampler.

That is, $T_\varepsilon(P)$ is the first step t, for which p_t is guaranteed to be at most ε away from the limit distribution p.

The spectral gap technique for bounding the burn-in period is applicable only to *reversible* Markov chains. A Markov chain is called *reversible*, if for all states $\mathrm{x}, \mathrm{y} \in \mathcal{U}$, $p(\mathrm{x})P(\mathrm{x},\mathrm{y}) = p(\mathrm{y})P(\mathrm{y},\mathrm{x})$. It can be shown that a reversible Markov chain is equivalent to a random walk on a weighted and undirected graph.

Let $n = |\mathcal{U}|$ and let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of P, ordered from largest to smallest by absolute value (i.e., $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$). The *spectral gap* of P is defined as the difference between its first and the second eigenvalues:

$$\alpha(P) = |\lambda_1| - |\lambda_2|.$$

The following is a folklore result in Markov Chain theory (see, for example, [39]), which bounds the burn-in period in terms of the spectral gap:

Theorem 5. *Let P be the transition matrix of a reversible Markov chain whose limit distribution is p. Then, for every $\varepsilon > 0$,*

$$T_\varepsilon(P) \leq \frac{1}{\alpha(P)} \left(\ln \frac{1}{p_{\min}} + \ln \frac{1}{\varepsilon} \right),$$

where $p_{\min} = \min_{x \in \mathcal{U}} p(x)$.

That is, the larger the spectral gap, the faster the convergence to the limit distribution.

It can be shown that if P is reversible, then also P_{MH} and P_{MD} are reversible, and thus we can use the spectral gap technique to estimate the required burn-in periods of the MH and the MD samplers.

4.4.4 Efficient random walk sampling

The naive method for generating multiple independent samples from (a distribution which is close to) the limit distribution p of a Markov chain is to run a new random walk for each desired sample.

This incurs high overhead, if the required number of samples is large. It turns out that in some situations it is possible to use a *single* random walk to generate multiple samples.

Aldous [1] (with further improvements by Gillman [17] and Kahale [27]) proposed an efficient procedure for generating dependent samples that can be used to perform accurate *density estimations*.

Suppose $\mathcal{A} \subseteq \mathcal{U}$ is a subset of the state space. The *density* of $\mathcal{A}$ under a probability measure p is the quantity $p(\mathcal{A})$. For example, when p is the uniform distribution on $\mathcal{U}$, the density of $\mathcal{A}$ is the ratio $|\mathcal{A}|/|\mathcal{U}|$. Suppose that we are given an "oracle" procedure, which on input $\mathrm{x} \in \mathcal{U}$ can determine whether $\mathrm{x} \in \mathcal{A}$ or not, and that we would like to use this procedure to estimate the density of $\mathcal{A}$. This type of estimation is very popular. One example from our domain is the estimation of the relative overlap between two search engines.

Given a Markov chain P whose limit distribution is p, the most obvious method for estimating $p(\mathcal{A})$ would be to run n random walks, each for $T_\epsilon(P)$ steps, and thereby obtain n i.i.d. samples from (a distribution which is close to) p. The estimator for $p(\mathcal{A})$ would then be the fraction of the n samples that fall into $\mathcal{A}$.

Aldous's procedure is more efficient. Instead of running n walks, Aldous suggests running only a single walk of length $T_\epsilon(P)+n\frac{1}{\alpha(P)}$, and use the last $n\frac{1}{\alpha(P)}$ states visited as the samples, disregarding the first $T_\epsilon(P)$ steps as *sample delay*. As shown in [1, 17, 27], these $n\frac{1}{\alpha(p)}$ dependent samples can be used to produce an estimate for $p(\mathcal{A})$, which is as good as the estimate obtained from the n independent samples. Overall, the Aldous procedures saves a factor of $\log(1/p_{\min})$ in the number of random walk steps over the naive procedure.

5 Approximate Monte Carlo methods

All Monte Carlo methods assume that the trial distribution p is known, up to normalization. This assumption turns out to be very problematic in our setting, since the trial distributions we construct depend on unknown internal data of the search engine. An *approximate Monte Carlo method* employs an "approximate trial distribution" q rather than the true trial distribution p in the acceptance function calculations. The mismatch between the trial samples (that are generated from p) and the acceptance function (which is based on q) implies that the sampling distributions of the resulting procedures are no longer guaranteed to equal the target distribution π. To the best of our knowledge, no previous study addressed this scenario before.

We show that the sampling distribution of approximate rejection sampling and the limit distributions of approximate Metropolis-Hastings and of approximate Maximum Degree are all identical to some distribution π', for which we give a closed form formula. We then prove that π' is close to the target distribution π, as long as the trial distribution p and the approximate trial distribution q are similar. We also prove that the estimations generated by approximate importance sampling are close to the true values. All proofs are provided in Appendix A.

5.1 Approximate rejection sampling

Consider the following modified ("approximate") form of rejection sampling. The procedure is given the following three oracle procedures: (1) $\texttt{getSample}_p()$, which generates samples from p, (2) $\texttt{getWeight}_{\hat{\pi}}(\mathrm{x})$, which calculates an unnormalized form of the target distribution π; and (3) $\texttt{getWeight}_{\hat{q}}(\mathrm{x})$, which calculates an unnormalized form of an "approximate trial distribution" q. π, p, q are assumed to satisfy:

$$\mathrm{supp}(\pi) \subseteq \mathrm{supp}(p) \subseteq \mathrm{supp}(q).$$

The approximate rejection sampling procedure works exactly like the standard procedure, except that the acceptance function it uses is the following:

$$r'_{\mathrm{RS}}(\mathrm{x}) = \frac{\hat{\pi}(\mathrm{x})}{C\ \hat{q}(\mathrm{x})},$$

where $C \geq \max_{\mathrm{x} \in \mathrm{supp}(p)} \frac{\hat{\pi}(\mathrm{x})}{\hat{q}(\mathrm{x})}$ is an envelope constant.

The following theorem characterizes the sampling distribution of the approximate rejection sampling procedure and analyzes its sample complexity:

Theorem 6. *The sampling distribution of the approximate rejection sampling procedure is:*

$$\pi'(x) \ = \ \pi(x)\ \frac{\frac{p(x)}{q(x)}}{\mathbb{E}_\pi\left(\frac{p(X)}{q(X)}\right)}.$$

The expected number of samples from p needed to generate each sample from π' is:

$$\frac{C Z_{\hat{q}}}{Z_{\hat{\pi}}\ \ \mathbb{E}_\pi\left(\frac{p(X)}{q(X)}\right)}.$$

The following proposition shows that as long as the trial distribution p and the approximate trial distribution q are "similar" relative to the target distribution π (in the sense that the ratio $p(\mathrm{X})/q(\mathrm{X})$ has low variance when X is chosen according to π), then the sampling distribution of approximate rejection sampling is close to the target distribution:

Proposition 7.

$$||\pi' - \pi|| = \frac{1}{2}\,\mathrm{ndev}_\pi\left(\frac{p(X)}{q(X)}\right).$$

5.2 Approximate Importance Sampling

The approximate importance sampling procedure assumes oracle access to the approximate trial distribution q. π, p, q are assumed to satisfy:

$$\mathrm{supp}(\pi) \subseteq \mathrm{supp}(p) \subseteq \mathrm{supp}(q).$$

The approximate importance sampling procedure works exactly like the standard importance sampling procedure, except that the following approximate importance ratios are used:

$$w'(\mathrm{x}) = \frac{\hat{\pi}(\mathrm{x})}{\hat{q}(\mathrm{x})}.$$

The following theorem shows that as long as the ratio $p(\mathrm{X})/q(\mathrm{X})$ is uncorrelated with the function $f(\mathrm{X})$ whose expectation we need to estimate, then the estimate produced by approximate importance sampling is close to the desired parameter:

Theorem 8. *Let*

$$\hat{\mu}' = \frac{\hat{\mu}'_1}{\hat{\mu}'_2} = \frac{\frac{1}{n}\sum_{i=1}^{n} f(X_i)w'(X_i)}{\frac{1}{n}\sum_{i=1}^{n} w'(X_i)}$$

be the estimator produced by the approximate importance sampling procedure for the parameter $E_\pi(f(X))$. Then,

$$\frac{\mathbb{E}_p(\hat{\mu}'_1)}{\mathbb{E}_p(\hat{\mu}'_2)} = \mathbb{E}_\pi(f(X)) + \frac{\mathrm{cov}_\pi\left(f(X), \frac{p(X)}{q(X)}\right)}{E_\pi\left(\frac{p(X)}{q(X)}\right)}.$$

5.3 Approximate Metropolis-Hastings

Next, we discuss an approximate variant of the Metropolis-Hastings algorithm. Like in approximate rejection sampling, we assume oracle access to an approximate trial distribution q. We assume that $\mathrm{supp}(q) = \mathrm{supp}(p) = \mathrm{supp}(\pi) = \mathcal{U}$.

We also need to assume that the base Markov chain P is reversible (i.e., $p(\mathrm{x})P(\mathrm{x},\mathrm{y}) = p(\mathrm{y})P(\mathrm{y},\mathrm{x})$, for all $\mathrm{x},\mathrm{y} \in \mathcal{U}$). When P is reversible, the acceptance function of the standard Metropolis-Hastings algorithm can be rewritten as follows:

$$r_{\text{MH}}(\mathrm{x},\mathrm{y}) \;=\; \min\left\{\frac{\pi(\mathrm{y})\ P(\mathrm{y},\mathrm{x})}{\pi(\mathrm{x})\ P(\mathrm{x},\mathrm{y})},\ 1\right\} \;=\; \min\left\{\frac{\pi(\mathrm{y})\ p(\mathrm{x})}{\pi(\mathrm{x})\ p(\mathrm{y})},\ 1\right\}.$$

The approximate Metropolis-Hastings procedure is identical to the standard procedure, except that it uses the following acceptance function:

$$r'_{\text{MH}}(\mathrm{x},\mathrm{y}) \;=\; \min\left\{\frac{\pi(\mathrm{y})\ q(\mathrm{x})}{\pi(\mathrm{x})\ q(\mathrm{y})},\ 1\right\}.$$

Note that since $\frac{\hat{\pi}(\mathrm{y})}{\hat{\pi}(\mathrm{x})} = \frac{\pi(\mathrm{y})}{\pi(\mathrm{x})}$ and $\frac{\hat{q}(\mathrm{x})}{\hat{q}(\mathrm{y})} = \frac{q(\mathrm{x})}{q(\mathrm{y})}$, this acceptance function can be computed using oracle access to unnormalized forms of π and q.

The following theorem shows that the resulting Markov chain is ergodic and that its unique limit distribution is π'.

Theorem 9. *Let P'_{MH} be the transition matrix of the approximate Metropolis-Hastings algorithm. Then, P'_{MH} forms an ergodic Markov chain and its unique limit distribution is π'.*

It follows from Proposition 7 that the limit distribution of the approximate MH random walk is close to the target distribution π as long as p and q are similar relative to π.

5.4 Approximate Maximum Degree

As before, we assume oracle access to an approximate trial distribution q and that $\mathrm{supp}(q) = \mathrm{supp}(p) = \mathrm{supp}(\pi) = \mathcal{U}$. This time we do not need to assume that the base Markov chain P is reversible.

Approximate MD is identical to the standard MD, except that the following modified acceptance function is used:

$$r'_{\mathrm{MD}}(\mathrm{x}) \;=\; \frac{\hat{q}(\mathrm{x})}{C\;\hat{\pi}(\mathrm{x})}, \quad \text{where} \quad C \geq \max_{\mathrm{x}\in\mathcal{U}} \frac{\hat{q}(\mathrm{x})}{\hat{\pi}(\mathrm{x})}.$$

The following theorem shows that the resulting Markov chain is ergodic and that its unique limit distribution equals π'.

Theorem 10. *Let P'_{MD} be the transition matrix of the approximate Maximum Degree procedure. Then, P'_{MD} forms an ergodic Markov chain. The unique limit distribution of P'_{MD} is π'.*

It follows from Proposition 7 that the limit distribution of the approximate MD random walk is close to the target distribution π as long as p and q are close relative to π.

6 Formal framework

In this section we lay out the formal framework for the design and analysis of search engine samplers.

6.1 Search engines

Definition 11 (Search engine). A *search engine* is a 4-tuple $\langle \mathcal{D}, \mathcal{Q}, \mathsf{results}(\cdot), k \rangle$, where:

1. $\mathcal{D}$ is the document corpus indexed. Documents are assumed to have been pre-processed (e.g., they may be truncated to some maximum size limit).
2. $\mathcal{Q}$ is the space of queries supported by the search engine.
3. $\mathsf{results}(\cdot)$ is a mapping that maps every query $\mathrm{q} \in \mathcal{Q}$ to an ordered sequence of documents, called *results*. The *cardinality* of q is the number of results: $\mathrm{card}(\mathrm{q}) = |\mathsf{results}(\mathrm{q})|$.
4. k is the *result limit*. Only the top k results are actually returned to the user via the search engine's public interface.

A query q is said to be *overflowing*, if $\mathrm{card}(\mathrm{q}) > k$. It is said to be *underflowing*, if $\mathrm{card}(\mathrm{q}) = 0$. If q neither overflows nor underflows, it is called *valid*.

A document x *matches* a query q, if $\mathrm{x} \in \mathsf{results}(\mathrm{q})$. The set of queries that a document x matches is denoted $\mathsf{queries}(\mathrm{x})$.

6.2 Search engine samplers

Definition 12 (Search engine sampler). Let π be a target distribution on a document corpus $\mathcal{D}$ indexed by a search engine. A *search engine sampler* is a randomized procedure, which is given access to three "oracle" procedures:

1. `getWeight`$_{\hat{\pi}}$(x): returns the unnormalized weight of an instance x $\in \mathcal{D}$ under the target distribution π.
2. `getResults`(q): returns the top k results from the search engine on the query q.
3. `getDocument`(x): returns the HTTP header and the content of the document x.

Each invocation of the sampler outputs a random document X from the corpus $\mathcal{D}$. The distribution of the sample X is called the *sampling distribution* and is denoted by η. Successive invocations of the sampler produce independent samples from η.

If the unnormalized form of π is independent of the corpus $\mathcal{D}$ (e.g., when π is the uniform distribution and $\hat{\pi}(\mathrm{x}) = 1$ for all x), then the same sampler can be used to sample from different search engines. All that needs to be changed is the implementation of the procedure `getResults`(q).

When the sampling distribution η of the sampler equals exactly the target distribution π, then the sampler is said to be *perfect*. Otherwise, it is *biased*. We discuss below the two main metrics for measuring the quality of a sampler w.r.t. a given target distribution: the *sampling recall* and the *sampling bias*.

Not all documents in $\mathcal{D}$ are practically reachable via the public interface of the search engine. Some pages have no text content and others have very low static rank, and thus formulating a query that returns them as one of the top k results may be impossible. Thus, search engine samplers usually generate samples only from large subsets of $\mathcal{D}$ and not from the whole corpus $\mathcal{D}$. The *sampling recall* of a sampler with target π and sampling distribution η is defined as $\pi(\mathrm{supp}(\eta))$. For instance, when π is the uniform distribution, the sampling recall is $|\mathrm{supp}(\eta)|/|\mathcal{D}|$, i.e., the fraction of documents that the sampler can actually return as samples. Ideally, we would like the recall to be as close to 1 as possible. Note that even if the recall is lower than 1, but $\mathrm{supp}(\eta)$ is sufficiently representative of $\mathcal{D}$, then estimators that use samples from $\mathrm{supp}(\eta)$ can produce accurate estimates of parameters of the whole corpus $\mathcal{D}$.

Since samplers sample only from large subsets of $\mathcal{D}$ and not from $\mathcal{D}$ in its entirety, it is unfair to measure the bias of a sampler directly w.r.t. the target distribution π. Rather, we measure the bias w.r.t. the distribution π *restricted* to $\mathrm{supp}(\eta)$. Formally, let $\pi_{\mathrm{supp}(\eta)}$ be the following distribution on $\mathrm{supp}(\eta)$:

$$\pi_{\mathrm{supp}(\eta)}(\mathrm{x}) = \frac{\pi(\mathrm{x})}{\pi(\mathrm{supp}(\eta))}, \quad \forall \mathrm{x} \in \mathrm{supp}(\eta).$$

The *sampling bias* of the sampler is defined as the total variation distance between η and $\pi_{\mathrm{supp}(\eta)}$:

$$||\eta - \pi_{\mathrm{supp}(\eta)}|| = \frac{1}{2} \sum_{\mathrm{x} \in \mathrm{supp}(\eta)} |\eta(\mathrm{x}) - \frac{\pi(\mathrm{x})}{\pi(\mathrm{supp}(\eta))}|.$$

For example, if a sampler generates truly uniform samples from a subset $\mathcal{D}'$ of $\mathcal{D}$ that constitutes 80% of $\mathcal{D}$, then its sampling recall is 0.8 and its sampling bias is 0.

The two most expensive resources of a search engine sampler are: (1) the amount of queries submitted to the search engine; and (2) the amount of additional web pages fetched. Search engine queries and web page fetches consume significant amount of time and require network bandwidth. In addition, the rate at which a sampler can submit queries to the search engine is usually very restricted, since search engines impose hard daily limits on the number of queries they accept from any single user. We thus measure the complexity of search engine samplers in terms of their *query cost* (expected number of calls to the subroutine `getResults()` per sample generated) and their *fetch cost* (expected number of calls to the subroutine `getDocument()` per sample generated).

6.3 Query pools

Consider a search engine $\mathcal{S}$, whose corpus is $\mathcal{D}$ and whose query space is $\mathcal{Q}$.

Definition 13 (Query pool). A *query pool* is a fragment $\mathcal{P} \subseteq \mathcal{Q}$ of the query space.

A query pool may be specified either *explicitly* as a set of queries, e.g.,

$$\mathcal{P} = \{\text{[Java software]}, \text{["Michael Jordan" -basketball]}, \text{[Car OR Automobile]}\},$$

or *implicitly*, e.g.,

$$\mathcal{P} = \text{All single-term queries.}$$

Note that in the latter case in order to transform $\mathcal{P}$ into an explicit form, we need a list of all the terms that occur in the corpus $\mathcal{D}$. All the samplers we consider in this paper fix some query pool $\mathcal{P}$ and use only queries from $\mathcal{P}$ in order to generate the sample documents from $\mathcal{D}$.

Queries-documents graph Every query pool $\mathcal{P}$ naturally induces a bipartite graph $B_{\mathcal{P}}$ on $\mathcal{P} \times \mathcal{D}$. Its left side consists of all queries in $\mathcal{P}$ and its right side consists of all documents in $\mathcal{D}$. A query $q \in \mathcal{P}$ is connected to a document $x \in \mathcal{D}$ if and only if $x \in \mathsf{results}(q)$.[2]

The *cardinality* of a query q, denoted card(q), is its degree in $B_{\mathcal{P}}$. This is exactly the number of documents in the result set of the query. The cardinality of a set of queries $\mathcal{P}' \subseteq \mathcal{P}$ is defined as:

$$\text{card}(\mathcal{P}') = \sum_{q \in \mathcal{P}'} \text{card}(q).$$

For a document x, we denote by $\mathsf{queries}_{\mathcal{P}}(x)$ the set of its neighbors in $B_{\mathcal{P}}$. These are exactly all the queries in $\mathcal{P}$ that x matches. For example, if $\mathcal{P}$ is the pool of all single term queries, then $\mathsf{queries}_{\mathcal{P}}(x)$ is the set of all distinct terms that occur in the text of x. The *degree* of x is: $\deg_{\mathcal{P}}(x) = |\mathsf{queries}_{\mathcal{P}}(x)|$. The degree of a set of documents $\mathcal{D}' \subseteq \mathcal{D}$ is defined as:

$$\deg_{\mathcal{P}}(\mathcal{D}') = \sum_{x \in \mathcal{D}'} \deg_{\mathcal{P}}(x).$$

[2] A similar graph was suggested by Davison [13] in a different context.

Note that $\text{card}(\mathcal{P})$ is the sum of the degrees of all nodes on the left side of $B_{\mathcal{P}}$, while $\deg_{\mathcal{P}}(\mathcal{D})$ is the sum of the degrees of all nodes on the right side of $B_{\mathcal{P}}$. Both sums equal to the number of edges in $B_{\mathcal{P}}$, and we thus obtain the following result:

Proposition 14. *Let $\mathcal{P}$ be any query pool. Then,* $\text{card}(\mathcal{P}) = \deg_{\mathcal{P}}(\mathcal{D})$.

Recall We say that a query pool $\mathcal{P}$ *covers* a document x, if $\deg_{\mathcal{P}}(\text{x}) > 0$. That is, at least one query in $\mathcal{P}$ returns x as a result. Let $\mathcal{D}_{\mathcal{P}}$ be the collection of documents covered by $\mathcal{P}$. Note that a sampler that uses only queries from $\mathcal{P}$ can never reach documents outside $\mathcal{D}_{\mathcal{P}}$.

For a distribution π on $\mathcal{D}$, the *recall of $\mathcal{P}$ w.r.t. π*, denoted $\text{recall}_{\pi}(\mathcal{P})$, is the probability that a random document selected from π is covered by $\mathcal{P}$. That is,

$$\text{recall}_{\pi}(\mathcal{P}) = \pi(\mathcal{D}_{\mathcal{P}}).$$

In the case π is the uniform distribution on $\mathcal{D}$, $\text{recall}_{\pi}(\mathcal{P}) = |\mathcal{D}_{\mathcal{P}}|/|\mathcal{D}|$.

Overflow and underflow probabilities Recall that a query q is *valid* if it neither overflows nor underflows. The set of valid queries $\text{q} \in \mathcal{P}$ is denoted $\mathcal{P}_+$ and the set of invalid queries is denoted $\mathcal{P}_-$. Given a distribution ϕ on $\mathcal{P}$, we define the *overflow probability* of ϕ, denoted $\text{ovprob}(\phi)$, to be the probability that a random query Q chosen from ϕ overflows:

$$\text{ovprob}(\phi) = \Pr_{\phi}(\text{card}(\text{Q}) > k).$$

Similarly, the *underflow probability* of ϕ, denoted $\text{unprob}(\phi)$, is the probability that Q underflows:

$$\text{unprob}(\phi) = \Pr_{\phi}(\text{card}(\text{Q}) = 0).$$

Local accessibility The samplers we use in this paper require "local accessibility" to the queries-documents graph $B_{\mathcal{P}}$. By that we mean that the sampler needs efficient implementations of the following procedures that compute incidences in the graph:

1. $\texttt{getIncidentDocs}_{\mathcal{P}}(\text{q})$: Given a query $\text{q} \in \mathcal{P}$, returns all documents that are incident to q in $B_{\mathcal{P}}$, i.e., $\mathsf{results}(\text{q})$.
2. $\texttt{getIncidentQueries}_{\mathcal{P}}(\text{x})$: Given a document $\text{x} \in \mathcal{D}$, returns all queries that are incident to x in $B_{\mathcal{P}}$, i.e., $\mathsf{queries}_{\mathcal{P}}(\text{x})$.

We next propose efficient implementations of the above procedures (other implementations may be possible too). The implementation of the first procedure is trivial: just submit q to the search engine and output all the results returned. The cost of this implementation is a single search engine query. It has one caveat, though: it is applicable only to non-overflowing queries. If the given query overflows, the procedure returns only the top k documents in the result set.

The implementation of the second procedure is slightly more tricky. In fact, only certain query pools, which we call "admissible", admit this implementation.

We say that a query pool is *admissible*, if we can compute $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$ *directly* from the content of x and without submitting queries to the search engine. If $\mathcal{P}$ is admissible, then the cost of computing $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$ is a single page fetch. We next argue that pools that consist solely of standard term/phrase queries (e.g., [java], ["Michael Jordan"]) are admissible. Pools that contain other types of queries, like complex Boolean queries or link queries (which ask for documents that contain a link to a given URL), may not be admissible.

Suppose q is a term/phrase query. That is, q corresponds to a term or a phrase t. A document x matches the query q if and only if x is indexed by the search engine under the term/phrase t. Suppose we know x is included in the corpus $\mathcal{D}$ of documents indexed by the search engine. Can we use the content of x alone to determine whether x is indexed under a given term/phrase t? Naively, the answer is yes: x is indexed under t if and only if t occurs in the text of x. Thus, by inspection of the content of x alone, one can determine whether x would belong to $\mathsf{results}(\mathrm{q})$. In practice, however, this may not be that simple. For instance, when a document x is too long, the search engine truncates it, and thus may not index x under terms that appear near its end. Conversely, search engines index documents under terms that do not occur at their text at all (e.g., anchor text terms). In general, the following factors may affect the choice of terms under which a document is indexed:

1. How the search engine pre-processes the document (e.g., whether it truncates the document).
2. How the search engine tokenizes the document (e.g., whether it ignores HTML tags).
3. How the search engine indexes the document (e.g., whether it filters out stopwords or whether it indexes also by anchor text terms).

Some of the above are not publicly available. Yet, most search engines follow standard IR methodologies and reverse engineering work (see, e.g., [7]) can be used to learn answers to the above questions. We therefore assume from now on that determining whether a document $\mathrm{x} \in \mathcal{D}$ matches a term/phrase query q can be done by inspecting the content of x alone.

Remark. In our experiments, though, we do not totally rely on this assumption. When we submit term/phrase queries to the search engine, we reject results that we would not have been determined as matching the query based on their text content alone.

Suppose now that $\mathcal{P}$ consists solely of term/phrase queries. Why is it admissible? Given a document x, we can compute $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$ as follows: we fetch x, enumerate all the terms/phrases that occur in x, and check which ones appear in $\mathcal{P}$. If $\mathcal{P}$ is given explicitly, then checking whether a given term/phrase appears in $\mathcal{P}$ is easy, assuming all the terms/phrases in $\mathcal{P}$ have been stored in a suitable dictionary data structure, like a hash table. If $\mathcal{P}$ is given in implicit form, then we simply have to check whether the predicate that defines $\mathcal{P}$ is satisfied by the given term/phrase.

We note that if $\mathcal{P}$ is given in implicit form, then pools that consist of even more complex Boolean term/phrase queries (e.g., "all two-term conjunctions" or "all three-term disjunctions") may be admissible too. On the other hand, implicit pools may be inadmissible, even if they consist solely of term/phrase queries. For example, if $\mathcal{P}$ is an explicit pool of term/phrase queries, then the implicit pool $\mathcal{P}_+$, which consists of all the terms/phrases in $\mathcal{P}$ that return between 1 and k results, is not admissible, because given a document x, we cannot know which of the terms/phrases that occur in x overflow and which do not (unless we submit them to the search engine).

7 Pool-based sampler

In this section we describe our pool-based (PB) sampler. The sampler assumes knowledge of an *explicit* and *admissible* query pool $\mathcal{P}$. Such a pool can be constructed by crawling a large corpus of web documents, such as the ODP directory [15], and collecting terms or phrases that occur in its pages. We can run the PB sampler with any such pool, yet the choice of the pool may affect the bias, the recall, and the query and fetch costs of the sampler. In the end of the section we provide analysis of the impact of the choice of the query pool on the performance of the PB sampler.

Let π be a target distribution on the corpus $\mathcal{D}$. We assume our sampler is given a black box procedure $\texttt{getWeight}_{\hat{\pi}}(\mathrm{x})$ that computes an unnormalized form $\hat{\pi}$ of π. We denote by $\mathrm{qcost}(\hat{\pi})$ and by $\mathrm{fcost}(\hat{\pi})$ the worst-case number of search engine queries and page fetches, respectively, the procedure uses to compute the weight $\hat{\pi}(\mathrm{x})$. As a running example, we think of π as the uniform distribution on $\mathcal{D}$. The unnormalized form we use is $\forall \mathrm{x}, \hat{\pi}(\mathrm{x}) = 1$, and thus $\mathrm{qcost}(\hat{\pi}) = \mathrm{fcost}(\hat{\pi}) = 0$.

The PB sampler does not directly generate samples from the target distribution π. Instead, it uses another sampler– the *degree distribution sampler*—that generates samples from the "document degree distribution" (see definition below). An unnormalized form of the document degree distribution can be efficiently computed. The PB sampler therefore applies a Monte Carlo method (e.g., rejection sampling) on the samples from the degree distribution in order to generate samples from π.

7.1 The outer procedure

Recall that the degree of a document $\mathrm{x} \in \mathcal{D}$ is the number of queries it matches:

$$\deg_{\mathcal{P}}(\mathrm{x}) = |\mathsf{queries}_{\mathcal{P}}(\mathrm{x})| = |\{\mathrm{q} \in \mathcal{P} \mid \mathrm{x} \in \mathsf{results}(\mathrm{q})\}|.$$

The distribution of documents by degree is called the *document degree distribution*:

$$d_{\mathcal{P}}(\mathrm{x}) = \frac{\deg_{\mathcal{P}}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathcal{D})}.$$

Note that the support of $d_{\mathcal{P}}$ is exactly $\mathcal{D}_{\mathcal{P}}$—the set of documents that are covered by $\mathcal{P}$. The document degree distribution has an unnormalized form, which is easy to compute:

$$\hat{d}_{\mathcal{P}}(\mathrm{x}) = \deg_{\mathcal{P}}(\mathrm{x}).$$

Recall that $\mathcal{P}$ is an admissible query pool, and thus computing $\deg_{\mathcal{P}}(\mathrm{x}) = |\mathsf{queries}_{\mathcal{P}}(\mathrm{x})|$ can be done by fetching only x and without submitting queries to the search engine.

Assume for the moment that we already have a sampler (DDSampler) that generates random documents sampled from the degree distribution $d_{\mathcal{P}}$ (we show in the next subsection how to construct such a sampler). Figure 6 shows the outer function of the PB sampler, which uses DDSampler as a subroutine.

The PB sampler applies rejection sampling with trial distribution $d_{\mathcal{P}}$ and target distribution π. The unnormalized weights used for document x are $\hat{\pi}(\mathrm{x})$ (which is computed by calling the

```
1: Function PBSampler(SE,C)
2:   while (true) do
3:     X := random document generated by DDSampler(SE)
4:     toss a coin whose heads probability is π̂(X) / (C deg_P(X))
5:     if (coin comes up heads)
6:       break
7:   return X
```

Figure 6: The outer procedure of the PB sampler.

$\texttt{getWeight}_{\hat{\pi}}(\mathrm{x})$ procedure) and $\hat{d}_{\mathcal{P}}(\mathrm{x}) = \deg_{\mathcal{P}}(\mathrm{x})$. Recall that the latter can be computed by a single page fetch. An envelope constant C satisfying

$$C \geq \max_{\mathrm{x} \in \mathcal{D}_{\mathcal{P}}} \frac{\hat{\pi}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathrm{x})}$$

must be given to the PB sampler as input. In the case π is the uniform distribution on $\mathcal{D}$, $\hat{\pi}(\mathrm{x}) = 1$ for all $\mathrm{x} \in \mathcal{D}$ while $\deg_{\mathcal{P}}(\mathrm{x}) \geq 1$ for all $\mathrm{x} \in \mathrm{supp}(d_{\mathcal{P}}) = D_{\mathcal{P}}$. Therefore, in this case an envelope constant of $C = 1$ will do. The resulting acceptance probability (Line 4) is simply $1/\deg_{\mathcal{P}}(\mathrm{X})$.

We next analyze the recall and the bias of the PB sampler, under the assumption that DDSampler generates samples from $d_{\mathcal{P}}$:

Proposition 15. *Suppose the sampling distribution of DDSampler is exactly the degree distribution $d_{\mathcal{P}}$. Then, the sampling recall of the PB sampler is:*

$$\mathrm{recall}_{\pi}(\mathcal{P}) = \pi(\mathcal{D}_{\mathcal{P}})$$

and it is a perfect sampler, i.e., it has a sampling bias of 0.

Proof. Let η be the sampling distribution of the PB sampler. We first show that $\mathrm{supp}(\eta) = \mathcal{D}_{\mathcal{P}} \cap \mathrm{supp}(\pi)$. Clearly, $\mathrm{supp}(\eta) \subseteq \mathcal{D}_{\mathcal{P}}$, because the PB sampler cannot output documents that are not covered by $\mathcal{P}$. Also, $\mathrm{supp}(\eta) \subseteq \mathrm{supp}(\pi)$, because only documents that have non-zero probability under π can be accepted by the acceptance-rejection procedure. Therefore, $\mathrm{supp}(\eta) \subseteq \mathcal{D}_{\mathcal{P}} \cap \mathrm{supp}(\pi)$.

To show containment in the other direction, consider any document $\mathrm{x} \in \mathcal{D}_{\mathcal{P}} \cap \mathrm{supp}(\pi)$. This means that: (1) $\deg_{\mathcal{P}}(\mathrm{x}) > 0$; and (2) $\hat{\pi}(\mathrm{x}) > 0$. The first condition implies that x has a positive probability to be selected by DDSampler. The second condition implies that x has a positive probability to be accepted by the acceptance-rejection procedure. Therefore, x has an overall positive probability to be returned by the PB sampler and thus $\mathcal{D}_{\mathcal{P}} \cap \mathrm{supp}(\pi) \subseteq \mathrm{supp}(\eta)$.

We can now calculate the recall of the PB sampler:

$$\mathrm{recall}_{\pi}(PB) \;=\; \pi(\mathrm{supp}(\eta)) \;=\; \pi(\mathcal{D}_{\mathcal{P}} \cap \mathrm{supp}(\pi)) \;=\; \pi(\mathcal{D}_{\mathcal{P}}) \;=\; \mathrm{recall}_{\pi}(\mathcal{P}).$$

Next, we analyze the sampling bias of the PB sampler. To this end, we need to calculate the distance between η and the distribution obtained by restricting π to $\mathrm{supp}(\eta)$, i.e., $\pi_{\mathrm{supp}(\eta)}$. The

main thing to observe is that the unnormalized form $\hat{\pi}$ of π also gives an unnormalized form of $\pi_{\mathrm{supp}(\eta)}$. Let $Z_{\hat{\pi}}$ be the normalization constant of $\hat{\pi}$. Define

$$Z_{\hat{\pi}_{\mathrm{supp}(\eta)}} = Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}}).$$

Hence, for every $\mathrm{x} \in \mathrm{supp}(\eta)$, we have:

$$\pi_{\mathrm{supp}(\eta)}(\mathrm{x}) = \frac{\pi(\mathrm{x})}{\pi(\mathrm{supp}(\eta))} = \frac{\hat{\pi}(\mathrm{x})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}} \cap \mathrm{supp}(\pi))} = \frac{\hat{\pi}(\mathrm{x})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}})} = \frac{\hat{\pi}(\mathrm{x})}{Z_{\hat{\pi}_{\mathrm{supp}(\eta)}}}.$$

Therefore, $\hat{\pi}$ is indeed an unnormalized form of $\pi_{\mathrm{supp}(\eta)}$. So the right way to view the PB sampler is as a rejection sampling procedure with $\pi_{\mathrm{supp}(\eta)}$ (and not π) as the target distribution and with $d_{\mathcal{P}}$ as the trial distribution. Note that $\mathrm{supp}(\pi_{\mathrm{supp}(\eta)}) \subseteq \mathcal{D}_{\mathcal{P}} = \mathrm{supp}(d_{\mathcal{P}})$ and hence the necessary pre-condition of rejection sampling is met. It now follows from the analysis of rejection sampling that $\eta = \pi_{\mathrm{supp}(\eta)}$. □

We note that one could implement the PB sampler with other Monte Carlo methods as well. We chose to present here rejection sampling, due to its simplicity.

7.2 Degree distribution sampler

Next, we describe DDSampler—the sampler that samples documents from the degree distribution. To this end, we need to sample queries from the query pool according to the *query cardinality distribution*. Recall that the cardinality of a query is the number of documents in its result set. The distribution of queries by cardinality is defined as:

$$c_{\mathcal{P}}(\mathrm{q}) = \frac{\mathrm{card}(\mathrm{q})}{\mathrm{card}(\mathcal{P})}.$$

In Figure 7 we describe the degree distribution sampler. For the time being, we make two unrealistic assumptions: (1) There is a sampler QCSampler that samples queries from the cardinality distribution $c_{\mathcal{P}}$. (This seems unrealistic, because we do not know a priori the cardinalities of all the queries in $\mathcal{P}$, and so it is not clear how to sample queries proportionally to their cardinalities.) (2) No query in $\mathcal{P}$ overflows. (This is unrealistic, because it is not clear how to create a large explicit pool of queries that does not have even a single overflowing query.) We later show how to remove these assumptions.

1: **Function** DDSampler(SE)
2: Q := random query generated by QCSampler(SE)
3: submit Q to the search engine SE
4: results(Q) := results returned from SE
5: X := document chosen uniformly at random from results(Q)
6: return X

Figure 7: The degree distribution sampler.

The sampler is very similar to the Bharat-Broder sampler, except that it samples random queries proportionally to their cardinalities. Since no query overflows, all documents that match a query are included in its result set. It follows that the probability of a document to be sampled is proportional to the number of queries in $\mathcal{P}$ that it matches:

Proposition 16. *Suppose the sampling distribution of QCSampler is the query cardinality distribution and that $\mathcal{P}$ has no overflowing queries. Then, the sampling distribution of DDSampler is $d_{\mathcal{P}}$.*

Proof. Let p be the sampling distribution of DDSampler, and let X denote a random document selected by DDSampler. Let Q denote a random query chosen from the cardinality distribution $c_{\mathcal{P}}$. To calculate $p(\mathrm{x})$, we expand over all choices for Q:

$$p(\mathrm{x}) \;=\; \Pr_{p}(\mathrm{X}=\mathrm{x}) \;=\; \sum_{\mathrm{q}\in\mathcal{P}} \Pr_{p,c_{\mathcal{P}}}(\mathrm{X}=\mathrm{x}|\mathrm{Q}=\mathrm{q})\cdot\Pr_{c_{\mathcal{P}}}(\mathrm{Q}=\mathrm{q}).$$

Note that given $\mathrm{Q}=\mathrm{q}$, the probability that $\mathrm{X}=\mathrm{x}$ is 0 if $\mathrm{x}\notin\mathsf{results}(\mathrm{q})$ and is $1/\operatorname{card}(\mathrm{q})$ otherwise. Hence, the only terms left in the sum are ones that belong to $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$:

$$\begin{aligned}\sum_{\mathrm{q}\in\mathcal{P}} \Pr_{p,c_{\mathcal{P}}}(\mathrm{X}=\mathrm{x}|\mathrm{Q}=\mathrm{q})\cdot\Pr_{c_{\mathcal{P}}}(\mathrm{Q}=\mathrm{q}) \;&=\; \sum_{\mathrm{q}\in\mathsf{queries}_{\mathcal{P}}(\mathrm{x})} \frac{1}{\operatorname{card}(\mathrm{q})}\cdot\frac{\operatorname{card}(\mathrm{q})}{\operatorname{card}(\mathcal{P})}\\ &=\; \frac{|\mathsf{queries}_{\mathcal{P}}(\mathrm{x})|}{\operatorname{card}(\mathcal{P})} \;=\; \frac{\deg_{\mathcal{P}}(\mathrm{x})}{\operatorname{card}(\mathcal{P})}.\end{aligned}$$

Since $\operatorname{card}(\mathcal{P})=\deg_{\mathcal{P}}(\mathcal{D})$ (Proposition 14), then the LHS of the last expression equals $\frac{\deg_{\mathcal{P}}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathcal{D})} = d_{\mathcal{P}}(\mathrm{x})$. □

We next address the unrealistic assumption that none of the queries in $\mathcal{P}$ overflows. Rather than using $\mathcal{P}$, which is likely to have overflowing queries, we use the query pool $\mathcal{P}_+$ (recall that $\mathcal{P}_+$ is the set of valid queries in $\mathcal{P}$). $\mathcal{P}_+$ does not have any overflowing queries by definition.

In the next subsection we show an efficient implementation of QCSampler that generates samples from $c_{\mathcal{P}_+}$ (the cardinality distribution of $\mathcal{P}_+$) rather than from $c_{\mathcal{P}}$. Since $\mathcal{P}_+$ has no overflowing queries, then by Proposition 16, the sampling distribution of DDSampler in this case equals the degree distribution $d_{\mathcal{P}_+}$ induced by $\mathcal{P}_+$.

Let us now return to the outer function of the PB sampler. That function assumed DDSampler generates samples from $d_{\mathcal{P}}$. What happens if instead it generates samples from $d_{\mathcal{P}_+}$? Note that now there is a mismatch between the trial distribution used by the PB sampler (i.e., $d_{\mathcal{P}_+}$) and the unnormalized weights it uses (i.e., $\deg_{\mathcal{P}}(\mathrm{x})$).

One possible solution could be to try to compute the unnormalized weights of $d_{\mathcal{P}_+}$, i.e., $\hat{d}_{\mathcal{P}_+}(\mathrm{x}) = \deg_{\mathcal{P}_+}(\mathrm{x})$. However, this is impossible to do efficiently, because $\mathcal{P}_+$ is no longer an admissible query pool. Instead, we opt for a different solution: we leave the outer function of the PB sampler as is; that is, the trial distribution will be $d_{\mathcal{P}_+}$ but the unnormalized weights will remain those of $d_{\mathcal{P}}$ (i.e., $\deg_{\mathcal{P}}(\mathrm{x})$). This means that the PB sampler is in fact an approximate rejection sampling procedure, and we can thus use Theorem 6 to bound its sampling bias.

Theorem 18 below bounds the recall and the bias of the PB sampler. The upper bound on the bias is given in terms of a property of documents, which we call the *validity density*:

Definition 17 (Validity density). Let $\mathcal{P}$ be a query pool. The *validity density* of a document $\mathrm{x} \in \mathcal{D}_{\mathcal{P}}$ relative to $\mathcal{P}$ is:

$$\mathrm{vdensity}_{\mathcal{P}}(\mathrm{x}) = \frac{\deg_{\mathcal{P}_+}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathrm{x})}.$$

That is, the validity density of x is the fraction of valid queries among the queries from $\mathcal{P}$ that x matches. The bias of the PB sampler is bounded by half the normalized mean deviation of the validity density of documents, where documents are weighted by the target distribution. Hence, if all documents have more-or-less the same validity density, then we can expect the PB sampler to be accurate.

Theorem 18. *The sampling recall of the PB sampler is equal to:*

$$\mathrm{recall}_{\pi}(\mathcal{P}_+) = \pi(\mathcal{D}_{\mathcal{P}_+}).$$

The sampling bias of the PB sampler is at most:

$$\frac{1}{2}\,\mathrm{ndev}_{\pi_{\mathcal{P}_+}}(\mathrm{vdensity}_{\mathcal{P}}(X)),$$

where $\pi_{\mathcal{P}_+}$ is the restriction of π to $\mathcal{D}_{\mathcal{P}_+} \cap \mathrm{supp}(\pi)$.

For the proof, see Appendix B.

Why should the mean deviation (or, equivalently, the variance) of the validity density be small? Typically, the validity density of a document is related to the fraction of popular terms occurring in the document. The fraction of popular terms in documents written in the same language (or even in documents written in languages with similar statistical profiles) should be about the same.

Another factor that affects the variance of the validity density is the fraction of invalid queries among the queries in the pool. If invalid queries are rare, then the validity density of most documents will be close to 1, implying the variance of the validity density is small. This is formalized by Theorem 19 below.

To state the theorem, we first need to define a distribution over queries, with respect to which we measure the overflow probability. For every document $\mathrm{x} \in \mathcal{D}$, we define the "weight" of x to be its probability under the target distribution $\pi_{\mathcal{P}_+}$, i.e., $\pi_{\mathcal{P}_+}(\mathrm{x})$. The *weight* of a query is the sum of all the weights it "absorbs" from the documents it is connected to:

$$w_{\mathcal{P}}(\mathrm{q}) \;=\; \sum_{\mathrm{x} \in \mathsf{results}(\mathrm{q})} \frac{\pi_{\mathcal{P}_+}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathrm{x})}.$$

Note that each document x distributes its weight $\pi_{\mathcal{P}_+}(\mathrm{x})$ among all the queries it is connected to. Thus, its contribution to one query q is only $\pi_{\mathcal{P}_+}(\mathrm{x})/\deg_{\mathcal{P}}(\mathrm{x})$. It follows that the sum of all query weights equals the sum of all document weights:

$$w_{\mathcal{P}}(\mathcal{P}) \;=\; \sum_{\mathrm{q} \in \mathcal{P}} w_{\mathcal{P}}(\mathrm{q}) \;=\; \sum_{\mathrm{x} \in \mathcal{D}} \pi_{\mathcal{P}_+}(\mathrm{x}) \;=\; 1.$$

We can thus view $w_{\mathcal{P}}$ as a probability distribution over $\mathcal{P}$. We call this distribution the *query weight distribution*.

Theorem 19. *The sampling bias of the PB sampler is at most:*

$$\frac{\text{ovprob}(w_{\mathcal{P}})}{1 - \text{ovprob}(w_{\mathcal{P}})}.$$

That is, if the overflowing queries in the pool have a relatively low mass under the query weight distribution (i.e., they are few in number and they do not overflow by "much"), then the bias of the sampler is low. The proof of the theorem appears in Appendix B.

7.3 Cardinality distribution sampler

We are left to show how to efficiently sample queries from $\mathcal{P}$ according to the cardinality distribution $c_{\mathcal{P}_+}$. Sampling queries uniformly from $\mathcal{P}$ is easy, since we have $\mathcal{P}$ in explicit form. But how do we sample queries from $\mathcal{P}_+$ proportionally to their cardinalities? This seems impossible to do, because we do not know a priori which queries belong to $\mathcal{P}_+$ and what are the cardinalities of these queries.

Our most crucial observation is that an unnormalized form of $c_{\mathcal{P}_+}$ can be computed efficiently. Given a query $\text{q} \in \mathcal{P}$, a corresponding unnormalized weight is the following:

$$\hat{c}_{\mathcal{P}_+}(\text{q}) = \begin{cases} \text{card}(\text{q}), & \text{if } \text{card}(\text{q}) \leq k, \\ 0, & \text{if } \text{card}(\text{q}) > k. \end{cases}$$

$\hat{c}_{\mathcal{P}_+}(\text{q})$ can be computed by submitting q to the search engine and counting the number of matches it has.

Remark. Since we need to know card(q) exactly only when q does not overflow, then we can compute card(q) by physically counting the number of results returned by the search engine on the query q. We do not need to rely on the number of results reported by the search engine, which is notoriously inaccurate.

Now, since we know $c_{\mathcal{P}_+}$ in unnormalized form, we can apply rejection sampling with the uniform distribution on $\mathcal{P}$ as the trial distribution and with $c_{\mathcal{P}_+}$ as the target distribution. This will give us samples from $c_{\mathcal{P}_+}$.

The query cardinality sampler (QCSampler) is depicted in Figure 8. The sampler applies rejection sampling with the cardinality distribution $c_{\mathcal{P}_+}$ as the target distribution and with the uniform distribution on $\mathcal{P}$ (which we denote by $u_{\mathcal{P}}$) as the trial distribution. The unnormalized form used for the target distribution is $\hat{c}_{\mathcal{P}_+}$, as described above. The unnormalized form used for the trial distribution is:

$$\forall \text{q} \in \mathcal{P}, \quad \hat{u}_{\mathcal{P}}(\text{q}) = 1.$$

Since for every $\text{q} \in \mathcal{P}$, $\hat{c}_{\mathcal{P}_+}(\text{q}) \leq k$, then

$$\max_{\text{q} \in \mathcal{P}} \frac{\hat{c}_{\mathcal{P}_+}(\text{q})}{\hat{u}_{\mathcal{P}}(\text{q})} \leq k,$$

and thus the sampler uses the envelope constant $C = k$.

The following now follows directly from the correctness of rejection sampling:

```
1: Function QCSampler(SE)
2:    k := SE.result_limit
3:    while (true) do
4:       Q := uniformly chosen query from P
5:       submit Q to the search engine SE
6:       card(Q) := number of results returned from SE
7:       if (card(Q) > k)
8:          W := 0
9:       else
10:         W := card(Q)
11:      toss a coin whose heads probability is W/k
12:      if (coin comes up heads)
13:         break
14:   return Q
```

Figure 8: The cardinality distribution sampler.

Proposition 20. *The sampling distribution of QCSampler is* $c_{\mathcal{P}_+}$.

7.4 Cost analysis

We now analyze the query cost and the fetch cost of the PB sampler.

Theorem 21. *The query cost of the PB sampler is at most:*

$$\frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot (1 - \text{ovprob}(w_{\mathcal{P}}))} \cdot \left(\frac{|\mathcal{P}|}{|\mathcal{P}_+|} \cdot \frac{k}{\text{avg}_{q \in \mathcal{P}_+} \text{card}(q)} + \text{qcost}(\hat{\pi}) \right).$$

The fetch cost of the PB sampler is at most:

$$\frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot (1 - \text{ovprob}(w_{\mathcal{P}}))} \cdot (1 + \text{fcost}(\hat{\pi})).$$

The proof can be found in Appendix B.

The above expressions may seem hard to parse, so we would like to simplify and interpret them, at least for the case π is the uniform distribution on $\mathcal{D}$.

When π is uniform, $Z_{\hat{\pi}} = |\mathcal{D}|$ and $\pi(\mathcal{D}_{\mathcal{P}_+}) = |\mathcal{D}_{\mathcal{P}_+}|/|\mathcal{D}|$. Therefore, the term $Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+})$ is $|\mathcal{D}_{\mathcal{P}_+}|$. Also, an envelope constant $C = 1$ can be used in this case. It follows that

$$\frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+})} = \frac{\deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{|\mathcal{D}_{\mathcal{P}_+}|} = \text{avg}_{\text{x} \in \mathcal{D}_{\mathcal{P}_+}} \deg_{\mathcal{P}_+}(\text{x}).$$

Also, $\text{qcost}(\hat{\pi}) = \text{fcost}(\hat{\pi}) = 0$ in this case. Therefore, the query cost is:

$$\text{avg}_{\text{x} \in \mathcal{D}_{\mathcal{P}_+}} \deg_{\mathcal{P}_+}(\text{x}) \cdot \frac{1}{1 - \text{ovprob}(w_{\mathcal{P}})} \cdot \frac{|\mathcal{P}|}{|\mathcal{P}_+|} \cdot \frac{k}{\text{avg}_{\text{q} \in \mathcal{P}_+} \text{card}(\text{q})}.$$

Thus, the query cost is the product of four components: (1) The average degree of documents that are covered by the valid queries in $\mathcal{P}$; (2) The inverse of the "validity probability" of queries, when selected proportionally to their weights; (3) The ratio between the total number of queries in $\mathcal{P}$ and the valid queries in $\mathcal{P}$; and (3) The ratio between the maximum cardinality of valid queries (i.e., k) and the average cardinality of valid queries.

Similarly, the fetch cost in this case is:

$$\text{avg}_{\text{x}\in\mathcal{D}_{\mathcal{P}_+}} \deg_{\mathcal{P}_+}(\text{x}) \cdot \frac{1}{1-\text{ovprob}(w_{\mathcal{P}})}.$$

7.5 Choosing the query pool

We next review the parameters of the query pool that impact the PB sampler.

Pool's recall The sampler's recall equals the recall of $\mathcal{P}_+$—the pool of valid queries among the queries in $\mathcal{P}$. Therefore, we would like pools whose valid queries cover most of the documents in the corpus $\mathcal{D}$. In order to guarantee such high recall, the pool must consist of enough terms/phrases that are not too popular (and thus would not overflow), but yet almost every document in $\mathcal{D}$ contains at least one of them. We can obtain such a collection of terms/phrases by crawling a large corpus of web documents, such as the ODP directory.

Validity density deviation The bias of the PB sampler is bounded by the normalized mean deviation of the validity density of documents, where documents are weighted by the target distribution. That is, in order to keep the bias low, we need to make sure that all documents have roughly similar validity densities. If the query pool has few overflowing queries and the queries that do overflow do not overflow by much, then we should expect most documents to have a validity density that is close 1, implying that the variance of the validity density is small. Obtaining such a pool whose recall is still high may be tricky. A pool consisting of conjunctions or disjunctions of terms, for example, may be problematic, because such queries are likely to overflow. We thus opted for exact phrase queries. Our experiments indicate that phrases of length at least 5 are unlikely to overflow. If the phrases are collected from a sufficiently large and representative corpus, then the corresponding recall is still reasonable.

Average degree The query and fetch costs depend on the average degree of documents that are covered by the valid queries in the pool. Hence, we would like to find pools for which the degree of documents grows moderately with the document length. Exact phrase queries are a good example, because then the degree of documents grows linearly with the document length. Conjunctions or disjunctions of m terms are poor choices, because there the growth rate is exponential in m.

Overflow and underflow probabilities High density of overflowing queries in the pool has two negative effects: (1) it potentially increases the sampling bias of the sampler; and (2) it increases the query and fetch costs. High density of underflowing queries does not impact the sampling

bias, but may increase the query cost. We therefore would like to keep the overflow and underflow probabilities as small as possible.

Average cardinality The query cost depends also on the ratio between the maximum cardinality of valid queries (k) and the average cardinality of valid queries. We would like thus the average cardinality to be as close as possible to k. Of course, this may interfere with the overflow probability: if the cardinality is too high, many queries will simply overflow.

8 Random walk based sampler

We propose two variants of a random walk sampler. One is based on the Metropolis-Hastings algorithm and another on the Maximum Degree method. Both samplers perform a random walk on a virtual graph whose nodes are the documents indexed by the search engine.

Like the pool-based sampler, this sampler too selects its queries from a fixed admissible query pool $\mathcal{P}$. However, here the pool may be *implicit* rather than explicit, and thus does not require a pre-processing step for constructing the query pool.

Let π be a target distribution on the corpus $\mathcal{D}$. For technical reasons, we need to postulate in this section that $\text{supp}(\pi) = \mathcal{D}$, i.e., all documents have non-zero probability under π. As with the pool-based sampler, we assume access to an oracle procedure $\texttt{getWeight}_{\hat{\pi}}(\text{x})$ that computes an unnormalized form $\hat{\pi}$ of π. $\text{qcost}(\hat{\pi})$ and $\text{fcost}(\hat{\pi})$ denote, respectively, the worst-case query and fetch costs of this oracle procedure.

8.1 Document graph

The random walk sampler performs a random walk on a virtual graph $G_{\mathcal{P}}$, which we call the *document graph*.[3] $G_{\mathcal{P}}$ is obtained from the queries-documents graph $B_{\mathcal{P}}$ defined in Section 6.3. $G_{\mathcal{P}}$ is an undirected weighted graph whose vertex set is $\mathcal{D}$—the documents indexed by the search engine. Two documents x, y are connected by an edge in $G_{\mathcal{P}}$ if and only if they share a neighboring query q in the graph $B_{\mathcal{P}}$. The edge weight is the number of such shared neighbor queries, where each query is normalized by its cardinality. In other words, (x, y) is an edge if and only if $\mathsf{queries}_{\mathcal{P}}(\text{x}) \cap \mathsf{queries}_{\mathcal{P}}(\text{y}) \neq \emptyset$. The weight of the edge (x, y) is:

$$\text{weight}_{\mathcal{P}}(\text{x}, \text{y}) = \sum_{\text{q} \in \mathsf{queries}_{\mathcal{P}}(\text{x}) \cap \mathsf{queries}_{\mathcal{P}}(\text{y})} \frac{1}{\text{card}(\text{q})}.$$

The *degree* of a document x in $G_{\mathcal{P}}$ is defined as:

$$\deg_{G_{\mathcal{P}}}(\text{x}) \;=\; \sum_{\text{y} \in \mathcal{D}} \text{weight}_{\mathcal{P}}(\text{x}, \text{y}).$$

We next observe that the degree of a document x in the graph $G_{\mathcal{P}}$ coincides with its degree in the graph $B_{\mathcal{P}}$:

[3] The random walk is actually performed on $G_{\mathcal{P}_+}$, which is derived from the pool of valid queries in $\mathcal{P}$. See details below.

Proposition 22. *For every document $x \in \mathcal{D}$,*

$$\deg_{G_{\mathcal{P}}}(x) = |\mathsf{queries}_{\mathcal{P}}(x)| = \deg_{\mathcal{P}}(x).$$

Proof. For every triple $(\mathrm{x}, \mathrm{y}, \mathrm{q})$, where $\mathrm{x}, \mathrm{y} \in \mathcal{D}$ are documents and $\mathrm{q} \in \mathcal{P}$ is a query, define the predicate $A(\mathrm{x}, \mathrm{y}, \mathrm{q})$ to be 1 if and only if both x and y belong to $\mathsf{results}(\mathrm{q})$. Now, we use the predicate A to rewrite the degree of a document x:

$$\begin{aligned}
\deg_{G_{\mathcal{P}}}(\mathrm{x}) &= \sum_{\mathrm{y} \in \mathcal{D}} \mathrm{weight}_{\mathcal{P}}(\mathrm{x}, \mathrm{y}) \\
&= \sum_{\mathrm{y} \in \mathcal{D}} \sum_{\mathrm{q} \in \mathsf{queries}_{\mathcal{P}}(\mathrm{x}) \cap \mathsf{queries}_{\mathcal{P}}(\mathrm{y})} \frac{1}{\mathrm{card}(\mathrm{q})} \\
&= \sum_{\mathrm{y} \in \mathcal{D}} \sum_{\mathrm{q} \in \mathcal{P}} \frac{A(\mathrm{x}, \mathrm{y}, \mathrm{q})}{\mathrm{card}(\mathrm{q})} \\
&= \sum_{\mathrm{q} \in \mathcal{P}} \frac{1}{\mathrm{card}(\mathrm{q})} \sum_{\mathrm{y} \in \mathcal{D}} A(\mathrm{x}, \mathrm{y}, \mathrm{q}).
\end{aligned}$$

If $\mathrm{q} \in \mathsf{queries}_{\mathcal{P}}(\mathrm{x})$, then $\sum_{\mathrm{y} \in \mathcal{D}} A(\mathrm{x}, \mathrm{y}, \mathrm{q}) = |\mathsf{results}(\mathrm{q})| = \mathrm{card}(\mathrm{q})$. However, if $\mathrm{q} \notin \mathsf{queries}_{\mathcal{P}}(\mathrm{x})$, then $\sum_{\mathrm{y} \in \mathcal{D}} A(\mathrm{x}, \mathrm{y}, \mathrm{q}) = 0$. Hence, we have:

$$\sum_{\mathrm{q} \in \mathcal{P}} \frac{1}{\mathrm{card}(\mathrm{q})} \sum_{\mathrm{y} \in \mathcal{D}} A(\mathrm{x}, \mathrm{y}, \mathrm{q}) \;=\; \sum_{\mathrm{q} \in \mathsf{queries}_{\mathcal{P}}(\mathrm{x})} \frac{1}{\mathrm{card}(\mathrm{q})} \cdot \mathrm{card}(\mathrm{q}) \;=\; |\mathsf{queries}_{\mathcal{P}}(\mathrm{x})| \;=\; \deg_{\mathcal{P}}(\mathrm{x}).$$

□

8.2 Skeleton of the random walk sampler

The random walk sampler runs a random walk on the document graph $G_{\mathcal{P}_+}$ (like the pool-based sampler, it ignores invalid queries). The transition matrix P of a simple random walk on this graph is the following:

$$P(\mathrm{x}, \mathrm{y}) = \frac{\mathrm{weight}_{\mathcal{P}_+}(\mathrm{x}, \mathrm{y})}{\deg_{\mathcal{P}_+}(\mathrm{x})}.$$

That is, having visited a node x in the graph, a neighbor y is chosen proportionally to the weight of the edge connecting x and y. It can be shown that this random walk is a reversible Markov chain whose limit distribution is the document degree distribution $d_{\mathcal{P}_+}$ (recall definition from Section 7.1). In order to transform this simple random walk into a Markov chain that converges to the target distribution π, an MCMC algorithm is applied. In this paper we focus on the Metropolis-Hastings and the Maximum Degree methods.

The skeleton of the random walk sampler is described in Figure 9. The sampler runs a simple random walk on the graph $G_{\mathcal{P}_+}$ augmented with an acceptance-rejection procedure. Having visited a node X in the graph, the function sampleNeighbor selects a random neighbor Y with probability $P(\mathrm{X}, \mathrm{Y}) = \frac{\mathrm{weight}_{\mathcal{P}_+}(\mathrm{X}, \mathrm{Y})}{\deg_{\mathcal{P}_+}(\mathrm{X})}$ (the implementation of this function is described below). An acceptance-rejection procedure is then applied on X and Y in order to determine whether Y should be accepted

as the next step of the random walk. The choice of the acceptance function depends on the particular MCMC method used. The two acceptance functions we employ are:

$$r_{\text{MH}}(\text{x},\text{y}) \;=\; \min\left\{\frac{\pi(\text{y})\;P(\text{y},\text{x})}{\pi(\text{x})\;P(\text{x},\text{y})},\;1\right\} \;=\; \min\left\{\frac{\pi(\text{y})\;\deg_{\mathcal{P}_+}(\text{x})}{\pi(\text{x})\;\deg_{\mathcal{P}_+}(\text{y})},\;1\right\};\quad \text{and}$$

$$r_{\text{MD}}(\text{x}) \;=\; \frac{\hat{d}_{\mathcal{P}_+}(\text{x})}{C\;\hat{\pi}(\text{x})} \;=\; \frac{\deg_{\mathcal{P}_+}(\text{x})}{C\;\hat{\pi}(\text{x})},$$

where

$$C \geq \max_{\text{x}\in\mathcal{D}_{\mathcal{P}_+}} \frac{\deg_{\mathcal{P}_+}(\text{x})}{\hat{\pi}(\text{x})}$$

is an envelope constant.

```
1:Function RWSampler(SE, B, x_0)
2:  X := x_0
3:  for t = 1 to B do
4:    Y := sampleNeighbor(SE, X)
5:    if (accept(X,Y))
6:      X := Y
7:  return X

1:Function accept(x, y)
2:   compute r_MCMC(x, y)
3:   toss a coin whose heads probability is r_MCMC(x, y)
4:   return true if and only if coin comes up heads
```

Figure 9: Skeleton of the random walk sampler.

In order to implement the random walk sampler, we need to address four issues: (1) how to select the start node x_0? (2) how to set the length of the burn-in period B? (3) how to implement the neighbor sampling procedure? and (4) how to calculate the acceptance functions r_{MH} and r_{MD}?

8.3 Selecting the start node

The graph $G_{\mathcal{P}_+}$ is not necessarily connected. When we choose the start node x_0 from some connected component F of $G_{\mathcal{P}_+}$, then the random walk will never reach nodes outside F. This implies that the Markov chain we produce will not necessarily converge to the target distribution π, but rather to the distribution π_F obtained by restricting π to F:

$$\pi_F(\text{x}) \;=\; \frac{\pi(\text{x})}{\pi(F)},\quad \text{for all } \text{x}\in F.$$

Therefore, the recall of the sampler in this case will be at most $\pi(F)$. In order to maximize recall, we would like the start node to belong to the component that has the highest mass under π. We do not have any rigorous technique for making such a selection. On the other hand, we speculate that for sufficiently rich query pools, $G_{\mathcal{P}_+}$ is expected to have a giant connected component, and

thus almost any document chosen as a start node will do. Our experimental results (see Section 9.3) support this speculation, as the largest connected component of $G_{\mathcal{P}_+}$ in a small search engine that we built constituted close to 99% of the nodes.

8.4 Setting the burn-in period

As shown in Section 4.4.3, the main factor that determines the length of the burn-in period is the *spectral gap* of the underlying transition matrix. Thus, in order to set B, we need an estimate of the spectral gaps of the transition matrices P_{MH} and P_{MD} obtained by applying the MH and the MD methods, respectively, on the simple random walk on $G_{\mathcal{P}_+}$.

We experimentally estimated these gaps for a small search engine that we built. The results, discussed more thoroughly in Section 9.3, provide the following estimates:

$$\alpha(P_{\text{MD}}) \geq \frac{1}{20,000}, \qquad \alpha(P_{\text{MH}}) \geq \frac{1}{20,000}.$$

As the connectivity of $G_{\mathcal{P}_+}$ in larger search engines is expected to be similar to the connectivity of $G_{\mathcal{P}_+}$ in the small search engine, we expect similar bounds to be applicable also to random walks on real search engines. See more details in Section 9.3.

8.5 Sampling neighbors

We now address the problem of sampling neighbors according to the transition matrix $P(\mathrm{x},\mathrm{y})$. The naive method to select such a random neighbor would be the following: given a document x, find all valid queries $\mathrm{q} \in \mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})$ that match x, choose one of them at random, submit the query to the search engine, and then pick one of its results at random. The only problem with this algorithm is that we do not know how to compute the set $\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})$ efficiently, since $\mathcal{P}_+$ is not an admissible query pool.

The solution to the above problem emanates from the following observation: $\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})$ is a subset of $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$, which we can compute efficiently, since $\mathcal{P}$ is an admissible query pool. So we can simply select random queries from $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$ until hitting a valid query. This random valid query will be uniform in $\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})$. The neighbor sampling procedure, described in Figure 10, implements this idea.

The following proposition proves the correctness of the neighbor sampling procedure:

Proposition 23. *Let x be any document in $\mathcal{D}_{\mathcal{P}_+}$ and let Y be the random neighbor selected by the procedure sampleNeighbor, when given x as input. Then, for every neighbor y of x in the graph $G_{\mathcal{P}_+}$,*

$$\Pr(Y = y) = \frac{\text{weight}_{\mathcal{P}_+}(x, y)}{\deg_{\mathcal{P}_+}(x)}.$$

Proof. To calculate $\Pr(\mathrm{Y} = \mathrm{y})$, we expand over all possibilities for the random query Q chosen from

```
1: Function sampleNeighbor(SE, x)
2:   queries_P(x) := getIncidentQueries_P(x)
3:   while (true) do
4:     Q := query chosen uniformly from queries_P(x)
5:     submit Q to the search engine SE
6:     if (Q neither overflows nor underflows)
7:       break
8:   results(Q) := results returned from SE
9:   Y := document chosen uniformly at random from results(Q)
10:  return Y
```

Figure 10: The neighbor sampling procedure.

$\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})$. Obviously, $\Pr(\mathrm{Q}=\mathrm{q})=0$ for all $\mathrm{q}\notin\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})$.

$$\Pr(\mathrm{Y}=\mathrm{y}) \;=\; \sum_{\mathrm{q}\in\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})} \Pr(\mathrm{Y}=\mathrm{y}|\mathrm{Q}=\mathrm{q})\cdot\Pr(\mathrm{Q}=\mathrm{q}).$$

For fixed q and y, $\Pr(\mathrm{Y}=\mathrm{y}|\mathrm{Q}=\mathrm{q})=\frac{1}{\mathrm{card}(\mathrm{q})}$, if $\mathrm{q}\in\mathsf{queries}_{\mathcal{P}_+}(\mathrm{y})$, and $\Pr(\mathrm{Y}=\mathrm{y}|\mathrm{Q}=\mathrm{q})=0$, otherwise. $\Pr(\mathrm{Q}=\mathrm{q})=\frac{1}{|\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})|}=\frac{1}{\deg_{\mathcal{P}_+}(\mathrm{x})}$. Therefore,

$$\begin{aligned}
&\sum_{\mathrm{q}\in\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})} \Pr(\mathrm{Y}=\mathrm{y}|\mathrm{Q}=\mathrm{q})\cdot\Pr(\mathrm{Q}=\mathrm{q})\\
&\quad= \sum_{\mathrm{q}\in\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})\cap\mathsf{queries}_{\mathcal{P}_+}(\mathrm{y})} \frac{1}{\mathrm{card}(\mathrm{q})}\cdot\frac{1}{\deg_{\mathcal{P}_+}(\mathrm{x})}\\
&\quad= \frac{\mathrm{weight}_{\mathcal{P}_+}(\mathrm{x},\mathrm{y})}{\deg_{\mathcal{P}_+}(\mathrm{x})}.
\end{aligned}$$

□

8.6 Calculating the acceptance functions

Next, we address the issue of how to calculate the acceptance functions of the MH and the MD samplers.

The acceptance function of the MH algorithm is:

$$r_{\mathrm{MH}}(\mathrm{x},\mathrm{y}) \;=\; \min\left\{\frac{\pi(\mathrm{y})\;\deg_{\mathcal{P}_+}(\mathrm{x})}{\pi(\mathrm{x})\;\deg_{\mathcal{P}_+}(\mathrm{y})},\;1\right\}.$$

The acceptance function of the MD method is:

$$r_{\mathrm{MD}}(\mathrm{x}) \;=\; \frac{\deg_{\mathcal{P}_+}(\mathrm{x})}{C\;\hat{\pi}(\mathrm{x})},\qquad \text{where}\quad C\geq\max_{\mathrm{x}\in\mathcal{D}_{\mathcal{P}_+}}\frac{\deg_{\mathcal{P}_+}(\mathrm{x})}{\hat{\pi}(\mathrm{x})}.$$

The problem is that we cannot compute the degrees $\deg_{\mathcal{P}_+}(\mathrm{x})$ and $\deg_{\mathcal{P}_+}(\mathrm{y})$ efficiently, since $\mathcal{P}_+$ is not an admissible query pool. What we do instead is apply perturbed acceptance functions:

$$r'_{\mathrm{MH}}(\mathrm{x},\mathrm{y}) \;=\; \min\left\{\frac{\pi(\mathrm{y})\;\deg_{\mathcal{P}}(\mathrm{x})}{\pi(\mathrm{x})\;\deg_{\mathcal{P}}(\mathrm{y})},\;1\right\}$$

and

$$r'_{\mathrm{MD}}(\mathrm{x}) \;=\; \frac{\deg_{\mathcal{P}}(\mathrm{x})}{C'\;\hat{\pi}(\mathrm{x})}, \quad \text{where} \quad C' \geq \max_{\mathrm{x}\in\mathcal{D}_{\mathcal{P}_+}} \frac{\deg_{\mathcal{P}}(\mathrm{x})}{\hat{\pi}(\mathrm{x})}.$$

(Note that when π is the uniform distribution, C' should be an upper limit on the maximum degree of documents.)

$r'_{\mathrm{MH}}(\mathrm{x},\mathrm{y})$ and $r'_{\mathrm{MD}}(\mathrm{x})$ can be computed efficiently, because $\mathcal{P}$ is an admissible query pool. The problem is that now the acceptance functions and the base Markov chain P on which they are applied are mismatching, and thus the limit distributions are no longer guaranteed to equal the target distribution π. This scenario is the one captured by the approximate Metropolis-Hastings and the approximate Maximum Degree procedures, described in Section 5.

Before we analyze the sampling bias and sampling recall of the resulting samplers, let us identify the exact form of the target distribution, trial distribution, and approximate trial distribution employed by the approximate MH and MD procedures.

Let F be the connected component of $G_{\mathcal{P}_+}$ to which the start vertex x_0 of the random walk belongs. Let d_F be the restriction of the degree distribution $d_{\mathcal{P}_+}$ to F:

$$d_F(\mathrm{x}) \;=\; \frac{d_{\mathcal{P}_+}(\mathrm{x})}{d_{\mathcal{P}_+}(F)}, \quad \text{for all } \mathrm{x}\in F.$$

As the random walk can never reach nodes outside F, then d_F, rather than $d_{\mathcal{P}_+}$, is the limit distribution of the simple random walk on $G_{\mathcal{P}_+}$ that starts at $\mathrm{x}_0 \in F$. Therefore, the trial distribution is d_F.

Similarly, as the random walk can never reach nodes outside F, the real target distribution when applying the MH and the MD samplers with $\mathrm{x}_0 \in F$ is the restriction of π to F, i.e., π_F. Indeed, it can be easily verified that the restriction of the unnormalized form of π to F constitutes an unnormalized form of π_F.

Finally, the approximate trial distribution used by the two procedures is the restriction of the degree distribution $d_{\mathcal{P}}$ to F:

$$q_F(\mathrm{x}) \;=\; \frac{d_{\mathcal{P}}(\mathrm{x})}{d_{\mathcal{P}}(F)}, \quad \text{for all } \mathrm{x}\in F.$$

Since $\mathrm{supp}(\pi) = \mathcal{D}$, $\mathrm{supp}(d_{\mathcal{P}_+}) = \mathcal{D}_{\mathcal{P}_+}$, $\mathrm{supp}(d_{\mathcal{P}}) = \mathcal{D}_{\mathcal{P}}$, and $F \subseteq \mathcal{D}_{\mathcal{P}_+} \subseteq \mathcal{D}_{\mathcal{P}} \subseteq \mathcal{D}$, then $\mathrm{supp}(\pi_F) = \mathrm{supp}(d_F) = \mathrm{supp}(q_F) = F$. Therefore, π_F, d_F, q_F satisfy the requirements of the approximate MH and MD procedures. Furthermore, the simple random walk on F constitutes a reversible Markov chain, as $G_{\mathcal{P}_+}$ is an undirected graph. Therefore, the necessary pre-condition of the approximate MH procedure is met.

Applying Theorems 10 and 9, we know that the random walks performed by the approximate MH and MD procedures have the following limit distribution η:

$$\eta(\mathrm{x}) = \pi_F(\mathrm{x}) \frac{\frac{d_F(\mathrm{x})}{q_F(\mathrm{x})}}{\mathbb{E}_{\pi_F}\left(\frac{d_F(\mathrm{X})}{q_F(\mathrm{X})}\right)}.$$

The following theorem bounds the sampling bias and the sampling recall of the MH and MD samplers:

Theorem 24. *Let $\varepsilon > 0$. Suppose we run the MH sampler (resp., the MD sampler) with a burn-in period B that guarantees the approximate MH Markov chain (resp., approximate MD Markov chain) reaches a distribution, which is at distance of at most ε from the limit distribution. Then, the sampling bias of the MH sampler (resp., MD sampler) is at most:*

$$\frac{1}{2}\,\mathrm{ndev}_{\pi_F}(\mathrm{vdensity}_{\mathcal{P}}(X)) + \varepsilon.$$

The sampling recall of the MH sampler (resp., MD sampler) is at least:

$$\pi(F) \cdot (1 - \frac{1}{2}\,\mathrm{ndev}_{\pi_F}(\mathrm{vdensity}_{\mathcal{P}}(X)) - \varepsilon).$$

The proof appears in Appendix C.

8.7 Optimized MD sampler

As discussed in Section 4.4.2, the special form of the acceptance function of the MD sampler allows for an optimized implementation. Since the acceptance function depends only on the current state x and not on the proposed state y, then rather than selecting a new proposed neighbor Y every time the acceptance-rejection procedure is invoked, we can select the neighbor only once, after acceptance is achieved. Further optimization is possible by selecting a priori the number of steps the random walk is going to spend at the state x, without actually performing the iterative coin tosses.

In this spirit, we describe in Figure 11 an optimized version of the MD sampler. Note that the fact the proposed neighbor is chosen only once results in significant savings in the number of search engine queries made. We analyze these savings below.

8.8 Cost analysis

We now provide rough analysis of the query and fetch costs of the MH and MD samplers. Rigorous analysis is postponed to future work.

The costs are given in terms of the length of the burn-in period B. The basic analysis given is the same for both MH and MD. The actual costs may be different, though, due to differences in the required burn-in periods. We also provide analysis of the optimized MD sampler, which is much more efficient.

1: **Function** OptimizedMDSampler(SE, B, x_0, C)
2: X := x_0
3: $t = 0$
4: while ($t < B$) do
5: delay := generate a geometric random variable whose success parameter is: $\frac{\deg_{\mathcal{P}}(\mathrm{x})}{C' \, \hat{\pi}(\mathrm{x})}$
6: $t := t +$ delay
7: if ($t \geq B$) break
8: Y := sampleNeighbor(SE,X)
9: X := Y
10: $t := t + 1$
11: return X

Figure 11: The optimized Maximum Degree sampler.

Query cost Search engine queries are made in two places: (1) in the sampleNeighbor procedure; and (2) (possibly) in the $\texttt{getWeight}_{\hat{\pi}}$ procedure in order to compute the unnormalized target weights. The sampleNeighbor procedure is called B times. When called with a document x as input, the procedure repeatedly selects random queries from $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$ until hitting a valid query. Therefore, the expected number of queries made in such a call is:

$$\frac{|\mathsf{queries}_{\mathcal{P}}(\mathrm{x})|}{|\mathsf{queries}_{\mathcal{P}_+}(\mathrm{x})|} = \frac{\deg_{\mathcal{P}}(\mathrm{x})}{\deg_{\mathcal{P}_+}(\mathrm{x})} = \frac{1}{\mathrm{vdensity}(\mathrm{x})}.$$

In order to figure out the total number of queries made along the random walk by the sampleNeighbor procedure, we need to know the distributions of the random documents encountered along the walk. We do not know these distributions exactly, but we know that they tend to the limit distribution η. For the purpose of the cost analysis, therefore, we simply assume that they all equal η. In this case, the expected number of queries made by the sampleNeighbor procedure throughout the execution becomes:

$$B \cdot \mathbb{E}_{\eta}\left(\frac{1}{\mathrm{vdensity}(\mathrm{X})}\right).$$

As η is close to the target distribution π_F, then the number of such queries made is approximately equal to:

$$B \cdot \mathbb{E}_{\pi_F}\left(\frac{1}{\mathrm{vdensity}(\mathrm{X})}\right).$$

The procedure $\texttt{getWeight}_{\hat{\pi}}$ is called once for every candidate neighbor y. The number of such candidates selected is at most B. Each invocation of the procedure makes at most $\mathrm{qcost}(\hat{\pi})$ search engines queries. Therefore, the total number of queries made by $\texttt{getWeight}_{\hat{\pi}}$ is at most $B \cdot \mathrm{qcost}(\hat{\pi})$.

We conclude that the query cost of the MH and MD samplers is approximately at most:

$$B \cdot \left(\mathbb{E}_{\pi_F}\left(\frac{1}{\mathrm{vdensity}(\mathrm{X})}\right) + \mathrm{qcost}(\hat{\pi})\right).$$

In the case π is the uniform distribution, $\mathrm{qcost}(\hat{\pi}) = 0$, and thus the query cost becomes:

$$B \cdot \mathbb{E}_{\pi_F}\left(\frac{1}{\mathrm{vdensity}(\mathrm{X})}\right).$$

Fetch cost Also page fetches are made only in the sampleNeighbor procedure and in the $\texttt{getWeight}_{\hat{\pi}}$ procedure. In sampleNeighbor(x) only x is fetched, in order to compute $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$. In $\texttt{getWeight}_{\hat{\pi}}(\mathrm{x})$, at most $\mathrm{fcost}(\hat{\pi})$ pages are fetched in order to compute $\hat{\pi}(\mathrm{x})$. As each of the procedures is invoked B times, the fetch cost is at most:

$$B \cdot (1 + \mathrm{fcost}(\hat{\pi})).$$

When π is the uniform distribution, $\mathrm{fcost}(\hat{\pi}) = 0$, and thus the fetch cost is only B.

Cost analysis for the optimized MD sampler The number of calls to the sampleNeighbor and $\texttt{getWeight}_{\hat{\pi}}$ procedures made in the optimized MD sampler is much smaller than B. This translates into direct savings in the query and fetch costs.

An MD random walk can be viewed as performing a simple random walk on F and augmenting it with different "delays" at the different nodes visited along the walk. Let B' be the number of simple random walk steps performed during the MD random walk. Note that B' is not a fixed parameter, like B, but is rather a random variable, which depends on the delays made at the nodes encountered along the random walk. The number of calls to sampleNeighbor and to $\texttt{getWeight}_{\hat{\pi}}$ is B'. Thus, in order to estimate the query and fetch costs of the optimized MD sampler, we need to calculate the expectation of B'.

When visiting a node x, the delay is a geometric random variable whose expectation is the following:

$$\mathbb{E}(\mathrm{delay}(\mathrm{x})) \;=\; \frac{C'\hat{\pi}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathrm{x})}.$$

For example, when π is the uniform distribution, then $\hat{\pi}(\mathrm{x}) = 1$ and C' is an upper limit on the maximum degree. Therefore, the expected delay is approximately the ratio between the maximum degree and the degree of x.

In order to figure out the expectation of B', we need to know the distributions of the random documents encountered along the simple random walk on F. As before, we do not know these distributions, but we know that they approach the degree distribution d_F on F. We thus assume in the analysis below that the distributions of all these nodes is d_F. Hence, the expected delay of a random node encountered along the random walk is:

$$\begin{aligned}
\mathbb{E}_{d_F}(\mathrm{delay}(\mathrm{X})) &= \mathbb{E}_{d_F}\left(C'\frac{\hat{\pi}(\mathrm{X})}{\deg_{\mathcal{P}}(\mathrm{X})}\right) \\
&= \sum_{\mathrm{x}\in F} \frac{\deg_{\mathcal{P}_+}(\mathrm{x})}{\deg_{\mathcal{P}_+}(F)} \cdot C' \cdot \frac{Z_{\hat{\pi}_F} \cdot \pi_F(\mathrm{x})}{\deg_{\mathcal{P}}(\mathrm{x})} \\
&= C' \cdot \frac{Z_{\hat{\pi}_F}}{\deg_{\mathcal{P}_+}(F)} \cdot \mathbb{E}_{\pi_F}(\mathrm{vdensity}(\mathrm{X})).
\end{aligned}$$

The total number of steps performed by the random walk is B. B can be viewed as the sum of the delays of the B' nodes encountered during the simple random walk. The delays are assumed to be i.i.d. random variables with expectation $\mathbb{E}_{d_F}(\mathrm{delay}(\mathrm{X}))$. Thus, using Wald's identity, we obtain:

$$B \;\approx\; \mathbb{E}(B') \cdot \mathbb{E}_{d_F}(\mathrm{delay}(\mathrm{X})).$$

Therefore,

$$\mathbb{E}(B') \;\approx\; \frac{B}{\mathbb{E}_{d_F}(\text{delay}(\text{X}))} \;=\; B \cdot \frac{\deg_{\mathcal{P}_+}(F)}{C' \cdot Z_{\hat{\pi}_F}} \cdot \frac{1}{\mathbb{E}_{\pi_F}(\text{vdensity}(\text{X}))}.$$

As we've shown before, the expected number of search engine queries made at sampleNeighbor when called with document x as input is $1/\,\text{vdensity}(\text{x})$. The documents on which the B' invocations of sampleNeighbor are executed are roughly distributed according to d_F. Therefore, the expected number of queries made in each call is approximately:

$$\mathbb{E}_{d_F}\left(\frac{1}{\text{vdensity}(\text{X})}\right) \;=\; \sum_{\text{x}\in F} \frac{\deg_{\mathcal{P}_+}(\text{x})}{\deg_{\mathcal{P}_+}(F)} \cdot \frac{\deg_{\mathcal{P}}(\text{x})}{\deg_{\mathcal{P}_+}(\text{x})} \;=\; \frac{\deg_{\mathcal{P}}(F)}{\deg_{\mathcal{P}_+}(F)}.$$

Each call to $\texttt{getWeight}_{\hat{\pi}}$ makes at most $\text{qcost}(\hat{\pi})$ queries. We can therefore estimate the query cost of the optimized MD sampler as:

$$B \cdot \frac{\deg_{\mathcal{P}_+}(F)}{C' \cdot Z_{\hat{\pi}_F}} \cdot \frac{1}{\mathbb{E}_{\pi_F}(\text{vdensity}(\text{X}))} \cdot \left(\frac{\deg_{\mathcal{P}}(F)}{\deg_{\mathcal{P}_+}(F)} + \text{qcost}(\hat{\pi})\right).$$

When π is the uniform distribution, this expression can be further simplified. In this case, (1) C' is an upper limit on the maximum degree $\max_{\text{x}\in F} \deg_{\mathcal{P}}(\text{x})$; (2) $Z_{\hat{\pi}_F} = |F|$; and (3) $\text{qcost}(\hat{\pi}) = 0$. Therefore, the query cost simplifies to:

$$B \cdot \frac{\deg_{\mathcal{P}}(F)}{\max_{\text{x}\in F} \deg_{\mathcal{P}}(\text{x}) \cdot |F|} \cdot \frac{1}{\mathbb{E}_{\pi_F}(\text{vdensity}(\text{X}))} \;=\; B \cdot \frac{\text{avg}_{\text{x}\in F} \deg_{\mathcal{P}}(\text{x})}{\max_{\text{x}\in F} \deg_{\mathcal{P}}(\text{x})} \cdot \frac{1}{\mathbb{E}_{\pi_F}(\text{vdensity}(\text{X}))}.$$

This query cost is lower by a factor of roughly $\max_{\text{x}\in F} \deg_{\mathcal{P}}(\text{x})/\,\text{avg}_{\text{x}\in F} \deg_{\mathcal{P}}(\text{x})$ than the query cost of the standard MD sampler.

A similar analysis for the fetch cost of the optimized MD sampler shows that it is approximately:

$$B \cdot \frac{\deg_{\mathcal{P}_+}(F)}{C' \cdot Z_{\hat{\pi}_F}} \cdot \frac{1}{\mathbb{E}_{\pi_F}(\text{vdensity}(\text{X}))} \cdot (1 + \text{fcost}(\hat{\pi})).$$

9 Experimental results

We conducted four sets of experiments: (1) *pool measurements*: estimation of parameters of selected query pools; (2) *spectral gap estimations*: measurement of the spectral gaps of the transition matrices used by the random walk samplers; (3) *evaluation experiments*: evaluation of the bias of our samplers and the Bharat-Broder (BB) sampler; and (4) *exploration experiments*: measurements of real search engines.

9.1 Experimental setup

In order to conduct the first three sets of experiments, we built a home-made search engine over a corpus of 2.4 million documents obtained from the Open Directory Project (ODP) [15]. The ODP directory crawl consisted of 3 million pages, of which we kept only the ones that we could

successfully fetch and parse, that were in text, HTML, or pdf format, and that were written in English. Each page was given a serial id, stored locally, and indexed by single terms and phrases. Only the first 10,000 terms in each page were considered. Exact phrases were not allowed to cross boundaries, such as paragraph boundaries. We used static ranking by document id to rank query results. Different experiments used different values of the result limit k. See more details below.

In our exploration experiments, conducted in April-May 2006, we submitted 395,000 queries to Google, 448,000 queries to MSN Search, and 370,000 queries to Yahoo!. Due to legal restrictions on automatic queries, we used the Google, MSN, and Yahoo! Web Search APIs, which are, reportedly, served from older and smaller corpora than the corpora used to serve human users. These APIs are limited to submitting only a few thousands of queries a day, which limited the scale of the experiments we could perform.

The first three sets of experiments were performed on a dual Intel Xeon 2.8GHz processor workstation with 2GB RAM and two 160GB disks. The exploration experiments were conducted on seven machines of varying configurations.

9.2 Pool measurements

In the first set of experiments, we wanted to measure the pool parameters that impact the quality and the efficiency of our samplers. In order to get a variety of results, we measured four different query pools: a single terms pool and three pools of exact phrases of lengths 3, 5, and 7. (We measured only four pools, because each measurement required substantial disk space and running time.)

In order to construct the pools, we split the ODP data set into two parts: a *training set*, consisting of every fifth page (when ordered by id), and a *test set*, consisting of the rest of the pages. The pools were built only from the training data, but the measurements were done only on the test data. In order to determine whether a query is overflowing, we set a result limit of $k = 20$.

All our measurements were made w.r.t. the uniform target distribution. We measured the following parameters: (1) the pool's size (total number of queries); (2) the fraction of overflowing queries; (3) the fraction of underflowing queries; (4) the average cardinality of valid queries; (5) the recall of valid queries (i.e., $|\mathcal{D}_{\mathcal{P}_+}|/|\mathcal{D}|$); (6) the average degree of documents relative to valid queries (i.e., $\text{avg}_{\text{x}\in\mathcal{D}_{\mathcal{P}_+}} \deg_{\mathcal{P}_+}(\text{x})$); (7) the average degree of documents relative to all queries (i.e., $\text{avg}_{\text{x}\in\mathcal{D}_{\mathcal{P}_+}} \deg_{\mathcal{P}}(\text{x})$); (8) the normalized mean deviation of the validity density (i.e., $\text{ndev}_{\pi_{\mathcal{P}_+}}(\text{vdensity}(\text{X}))$); (9) the overflow probability of the query weight distribution (i.e., $\text{ovprob}(w_{\mathcal{P}})$). The results of our measurements are tabulated in Table 1.

The measurements show that fraction of overflowing queries, average document degree, normalized mean deviation of validity density, and overflow probability improve as phrase length increases, while fraction of underflowing queries, recall, and average cardinality get worse. The results indicate that a phrase length of 5 achieves the best tradeoff among the parameters. It has very few overflowing queries (about 0.5%), while maintaining a recall of about 89%. The fraction of overflowing queries among 3-term phrases is a bit too high (3%), while the recall of the 7-term phrases is way too low (about 64%). The fraction of overflowing queries among single terms is surprisingly small. The explanation is that many of the terms are misspellings, technical terms, or digit strings.

Parameter	Single terms	Phrases (3)	Phrases (5)	Phrases (7)
Pool's size	2.6M	97.5M	155.9M	151.1M
Fraction of overflowing queries	11.4%	3%	0.4%	0.1%
Fraction of underflowing queries	40.3%	56%	76.2%	82.1%
Average cardinality of valid queries	4.7	3.6	2.2	1.7
Recall of valid queries	61.9%	96.8%	88.6%	64.5%
Average document degree (valid queries)	5.1	77	47.1	37.1
Average document degree (all queries)	293.4	246.8	75.9	51.3
Normalized mean deviation of validity density	0.8	0.34	0.34	0.4
Overflow probability of query weight distribution	0.98	0.7	0.43	0.3

Table 1: Results of pool parameter measurements.

The overflow probability of single terms, though, is very high.

Since the ODP data set is presumably representative of the web, we expect most of these measurements to represent the situation on real search engines. The only exceptions are the fraction of overflowing and underflowing queries and the average cardinality of valid queries, which are distorted due to the small size of the ODP data set. We thus measured these parameters on the Yahoo! search engine. The results are given in Table 2. It is encouraging to see that for 5-term phrases, the fraction of overflowing queries remains relatively low, while the fraction of underflowing queries goes down dramatically. The elevated fraction of overflowing queries among 3-term phrases is more evident here.

Parameter	Single terms	Phrases (3)	Phrases (5)	Phrases (7)
Fraction of overflowing queries	49.3%	18%	3.3%	1%
Fraction of underflowing queries	5.4%	4.3%	10.3%	14.7%
Average cardinality of valid queries	104.1	107.6	47.9	20.6

Table 2: Pool parameter measurements on Yahoo!.

Assuming the above measurements represent the situation on real search engines, we can use Theorems 18 and 21 to derive estimates of the sampling bias, sampling recall, query cost, and fetch cost for the pool-based sampler. The results are given in Table 3. In the estimations we used the measurements on the Yahoo! search engine, and thus set $k = 1,000$.

Parameter	Single terms	Phrases (3)	Phrases (5)	Phrases (7)
Sampling bias	0.4	0.17	0.17	0.2
Sampling recall	61.9%	96.8%	88.6%	64.5%
Query cost	5407.4	3070	1996.6	3052
Fetch cost	255	256.7	82.6	53

Table 3: Estimated bias, recall, and cost of the pool-based sampler.

9.3 Spectral gap estimations

We wanted to estimate the burn-in periods of the two random walk samplers: the MH sampler and the MD sampler. By Theorem 5, the main factor that determines the burn-in period is the spectral gap of the random walk's transition matrix. In order to obtain an accurate estimate of the spectral gaps w.r.t. a real search engine SE, we would have had to construct the adjacency matrix of the document graph $G_{\mathcal{P}_+}$ corresponding to SE, transform this matrix into the two transition matrices P_{MH} and P_{MD}, and then estimate the spectral gaps of these matrices.[4] However, since we did not have full access to the corpus of real search engines, we could not explicitly construct the matrices P_{MH} and P_{MD}.

Our only option then was to estimate the spectral gaps of the matrices corresponding to our home-made ODP search engine. Since the ODP corpus is a diverse set of documents, presumably representing the whole web, we expect the document graph of the ODP search engine to have similar connectivity properties as the document graph of real search engines. Spectral gap is a "structural" property of a transition matrix, which depends only on the connectivity in the document graph, and not on the size of the graph. Therefore, estimations of the spectral gaps corresponding to the ODP search engine could be representative of the spectral gaps corresponding to real search engines.

In order to make the connectivity of the document graph of the ODP search engine as similar as possible to the connectivity of the document graphs in larger search engines, we set in this experiment $k = 80$ as the result limit and used a pool of 3-term exact phrases. These settings made the graph more dense, similarly to document graphs that are based on very large corpora.

We constructed the two matrices P_{MH} and P_{MD} corresponding to the largest connected component of the document graph. This component contained 98.7% of the documents in the corpus, resulting in two matrices of dimensions 2.37 million by 2.37 million.

Since the matrices were so large, we could not compute their eigenvalues exactly. The largest eigenvalue of any transition matrix is always 1, since the matrix is stochastic. We thus had to estimate only the second largest eigenvalue. To this end, we applied the power iteration method (cf. [18]). We obtained the following lower bounds on the two spectral gaps:

$$\alpha(P_{\text{MD}}) \geq \frac{1}{20,000}, \qquad \alpha(P_{\text{MH}}) \geq \frac{1}{20,000}.$$

We note that the actual gaps could be larger, yet figuring them out exactly would have required many more iterations of the power method. The fact the two bounds are identical does not mean that the actual spectral gaps of the two Markov chains are identical. In reality, one of them may be much higher than the other.

Plugging in the above estimates in the formula for the burn-in period (Theorem 5) and assuming $\pi_{\min} = \frac{1}{2 \cdot 10^{10}}$, as in major search engines, and $\varepsilon = 0.1$, we obtained:

$$T_\varepsilon(P_{\text{MH}}) = T_\varepsilon(P_{\text{MD}}) \leq \frac{1}{\alpha(T)} \left(\ln \frac{1}{\pi_{\min}} + \ln \frac{1}{\varepsilon} \right) \approx 520,000.$$

[4] In order to get more precise estimates of the required burn-in periods, the spectral gaps of the transition matrices P'_{MH} and P'_{MD}, rather than P_{MH} and P_{MD}, should have been calculated. This is left for future work.

We can now use the cost analysis from Section 8.8 to estimate the query costs of the MH and the MD samplers. To this end, we: (1) assume that $G_{\mathcal{P}_+}$ is connected (and thus $F = \mathcal{D}_{\mathcal{P}_+}$); (2) employ the estimates for $E_{\pi_{\mathcal{P}_+}}(\text{vdensity(X)}) = 1 - \text{ovprob}(w_{\mathcal{P}})$ and for $\text{avg}_{\text{x}\in\mathcal{D}_{\mathcal{P}_+}} \deg_{\mathcal{P}}(\text{x})$ derived from our pool measurements. Plugging in the numbers into the formulae given in Section 8.8, we expect the query cost of the MH sampler to be roughly $1,730,300$ queries per sample, while the query cost of the optimized MD sampler (executed with $C' = 10,000$) to be roughly $42,800$ queries per sample.

We conclude that both the MH sampler and the MD sampler are significantly less efficient than the pool-based sampler. The cost of the MH sampler is prohibitive. The cost of the optimized MD sampler is much lower, and thus this sampler might be of practical use, if the desired number of samples is small. Recall the big advantage of this approach over the pool-based sampler: it does not need any pre-processing step to create an explicit query pool.

We stress that the above cost estimates are based on our theoretical analysis, which is pessimistic by nature. Our evaluation experiments described below indicate that in practice it may be possible to run the random walks for far fewer steps than is required by the theoretical bounds, yet obtain good samples.

9.4 Evaluation experiments

In order to estimate the biases of our samplers and of the BB sampler, we ran them on the ODP search engine. In this controlled environment we could compare the sampling results against the real data.

The corpus of our ODP search engine consisted of the test set only. A result limit of $k = 5$ was used in order to have an overflow probability comparable to the one on Yahoo!.

We ran five samplers: (1) the PB sampler with rejection sampling; (2) the PB sampler with importance sampling; (3) the MH sampler; (4) the MD sampler; and (5) the BB sampler. All the samplers used a query pool of 5-term phrases extracted from the ODP training set. In order to have a common basis, we allowed each sampler to submit exactly 5 million queries to the search engine.

Running the random walk samplers with the burn-in period dictated by the spectral gap estimations would have been useless, since we would have gotten very few samples from the 5 million queries submitted. We therefore opted for a short burn-in period of 1,000 steps for the MH sampler and 10,000 steps for the MD sampler (these settings turned out to produce a comparable number of samples to what the PB sampler produced). This enabled us to evaluate whether the shortened burn-in period actually caused the random walk samplers to produce biased samples.

Since each sampler has a different query cost, the samplers produced varying number of samples using the 5 million queries. Table 4 shows the actual number of samples generated by each sampler. The BB sampler generated the most samples, yet these samples are highly biased.

In Figure 12, we show the distribution of samples by document size. We ordered the documents in the corpus by size, from largest to smallest, and split them into 10 equally sized deciles. Truly uniform samples should distribute evenly among the deciles. The results show the overwhelming difference between our samplers and the BB sampler. The BB sampler generated a huge number

Sampler	# of queries	# of samples
PB + Rejection Sampling	5M	3,276
PB + Importance Sampling	5M	319,682
RW-MH	5M	4,234
RW-MD	5M	7,761
BB	5M	1,129,820

Table 4: Number of samples generated by each sampler when run over the ODP search engine with 5 million queries.

of samples at the top decile (more than 50%!). Our samplers, on the other hand, had no or little bias. The two PB samplers essentially show no bias, while the RW samplers have small negative bias towards very short documents, possibly due to the shortened burn-in period.

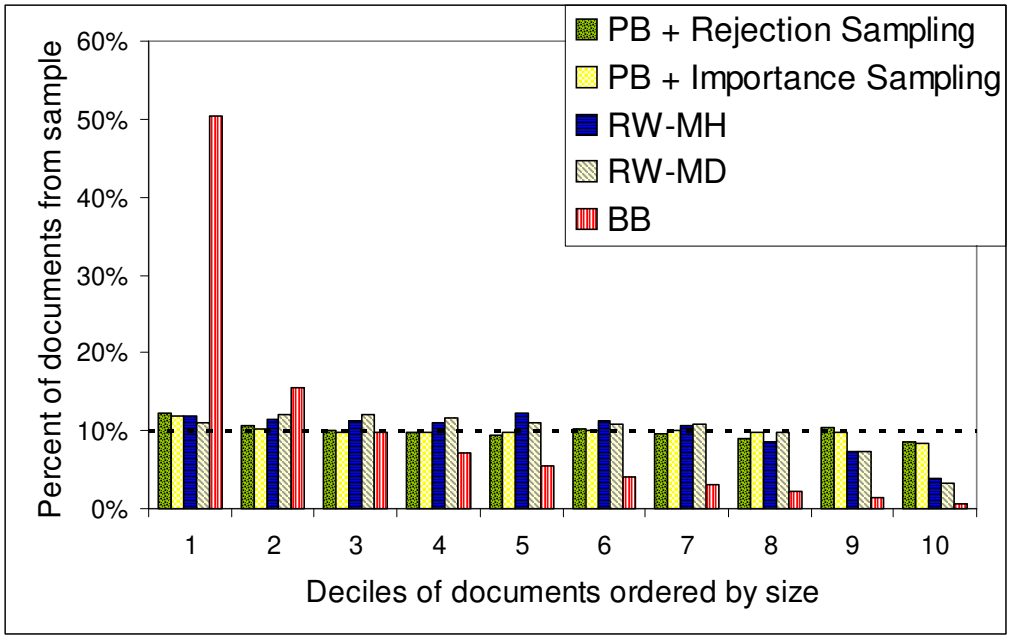

Figure 12: Distribution of samples by document size.

Figure 13 addresses bias towards highly ranked documents. We ordered the documents in the corpus by their static rank, from highest to lowest, and split into 10 equally sized deciles. The first decile corresponds to the most highly ranked documents. The results indicate that none of our samplers had any significant bias under the ranking criterion. Surprisingly, the BB sampler had bias towards the 4th, 5th, and 6th deciles. When digging into the data, we found that documents whose rank (i.e., id) belonged to these deciles had a higher average size than documents with lower or higher rank. Thus, the bias we see here is in fact an artifact of the bias towards long documents. A good explanation is that our 5-term exact phrases pool had a low overflow probability in the first place, so very few queries overflowed. Note that the ranking bias of the BB sampler is created only by overflowing queries.

We have several conclusions from the above experiments: (1) the 5-term phrases pool, which has

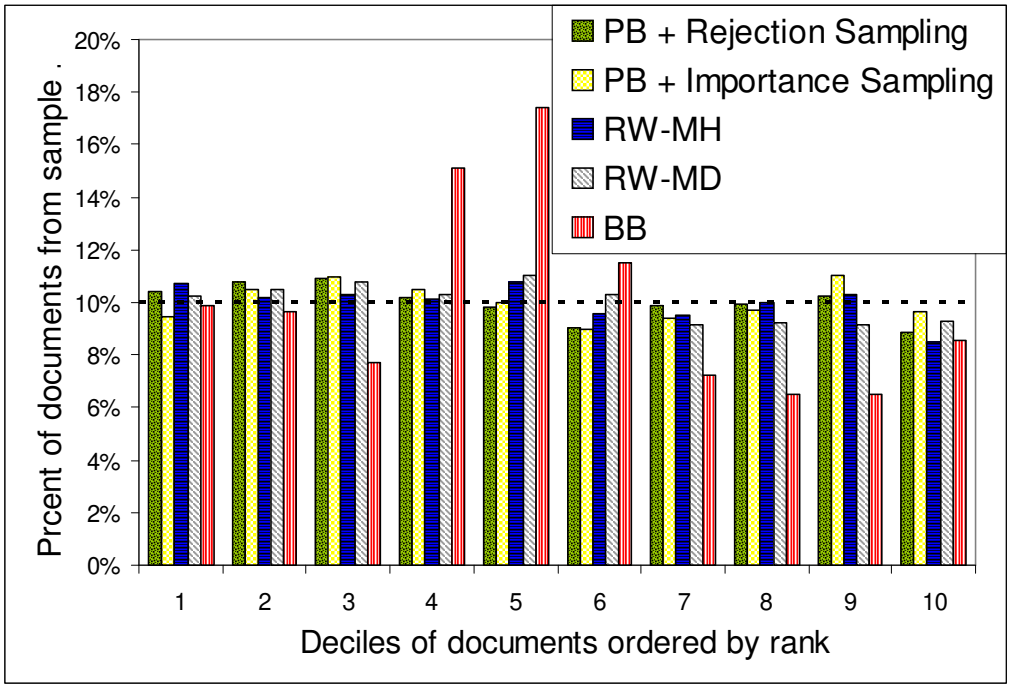

Figure 13: Distribution of samples by document rank.

small overflow probability, made an across-the-board improvement to all the samplers (including BB). This was evidenced by the lack of bias towards highly ranked documents. (2) The BB sampler suffers from a severe bias towards long documents, regardless of the query pool used. (3) Our pool-based samplers seem to give the best results, showing no bias in any of the experiments. (4) The random walk samplers have small negative bias towards short documents. Possibly by increasing the burn-in period of the random walk, this negative bias could be alleviated.

9.5 Exploration experiments

We used our most successful sampler, the PB sampler, to generate uniform samples from three major search engines: Google, MSN Search, and Yahoo!. Before describing the results of the experiment, we elaborate on the assumptions made in these experiments and on the procedure we used to test whether a given URL is indexed by a given search engine.

9.5.1 Assumptions and setup

The limit k on the number of returned results is 1,000 for Google and Yahoo!, and 250 for MSN Search. As a query pool, we used 5-term phrases extracted from English pages at the ODP data set. The pool contained over 600M phrases. During our experiments, we made the following assumptions: (1) The first $10,000$ terms in each document are indexed. (2) The corpora did not change during our experiments.

When calculating the cardinality of a query, we did not rely on the number of results reported by the search engine, since this number is notoriously inaccurate. Instead, we explicitly fetched the entire list of available results, and calculated its size.

When submitting queries to the search engines, we specifically indicated not to filter out duplicate results and not to perform any other filtering (language, domain, date, etc. . .). Note that duplicate results *are* filtered out by default, and thus our measurements do not directly reflect duplicate-free corpora but rather the complete "raw" corpora. We choose to make our measurements against the raw corpora in order to prevent different filtering mechanisms employed by search engines from biasing our measurements. Comparing duplicate-free corpora proves even more difficult because each search engine may choose a different "representative" from a set of duplicate pages, thus defeating our URL testing method.

Some documents are indexed by terms that do not appear in their content, e.g., according to anchor text. Since our samplers have access only to the terms that appear in the document, we rejected samples whose content did not contain the query terms. Similarly, to avoid bias due to index depth, we rejected sampled documents for which the query terms were not found among their first 10,000 terms.

In order to compute $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$, for a given document x, we first tried to retrieve the cached version of x from the search engine. If the cached page was not available, we downloaded the page from the web and computed $\mathsf{queries}_{\mathcal{P}}(\mathrm{x})$ according to the downloaded version. When any error was encountered while downloading the page, the corresponding sample was dropped.

Note that as a result of dropping some of the samples (either due to fetching errors or due to the absence of the query terms at the first 10,000 terms of the document), the query cardinalities, which were calculated assuming all query results are valid, were not always accurate. Since the frequency of sample drops was low, we speculate that this did not have any significant effect on our measurements.

9.5.2 URL testing

A basic ingredient in our exploration experiments was a procedure, which given a URL u and search engine SE, determines whether SE indexes u or not. Implementing such a procedure that produces accurate results, while interacting only with the public interface of the search engine, turns out to be tricky. The procedure we employed, which combines ideas from previous studies, is described below.

First, we brought the given URL into a canonical form, by converting hex-encoded characters (`%XX`) to a standard iso-8859-1 characters, converting double slash (`//`) to a single one, dropping `/index.html` from the URL's tail, etc.

For each URL tested, we submitted up to seven different queries to the search engine, in order to determine whether the URL is indexed by the search engine. The first query was the URL itself. The second query was "inurl:URL". The last five queries were phrases, from 8 to 14 terms long, extracted randomly from the first 10,000 terms of the document. We sequentially submitted these seven queries to the search engine until we found the URL among the first 100 results (after

canonization). If the URL was found by any of the queries, we assumed it is indexed by the search engine. Otherwise, it was assumed not to be indexed.

Obviously, this heuristic procedure is prone to some error. That is, its result is not guaranteed to correctly determine whether a URL is indexed or not.

If a URL was sampled from a search engine, but our procedure failed to find it in the same search engine, we dropped the sample. 5% of samples from Google, and less than 1% of samples from MSN Search and Yahoo! were dropped.

9.5.3 Corpus size

Table 5 tabulates the measured relative overlap of the Google, MSN Search, and Yahoo! corpora. We note that since our query pool consisted mostly of English language phrases, our results refer mainly to the English portions of these corpora.

Samples from ↓ indexed by →	Google	MSN Search	Yahoo!
Google		**37.2%**	**45.8%**
MSN Search	**51.2%**		**52.8%**
Yahoo!	**35.6%**	**30.9%**	

Table 5: Relative overlap of Google, MSN Search, and Yahoo!.

Using Liu's "rule of thumb" (see Section 4.3) and the Chernoff bound, we estimate the overlap results stated above to be within an absolute error of 4.5% from the real values with a confidence level of 95%. There could be an additional absolute error of up to 17%, due to the sampling bias of the PB sampler.

In Figure 14 we show the relative corpus sizes of the three search engines, as derived from the relative overlap estimates.

9.5.4 Top-level domain name distribution

Figure 15 shows the domain name distributions in the three corpora. Note that there are some minor differences between the different search engines. For example, Yahoo! has some bias towards the .com domain, while Google has negative bias towards the .info domain.

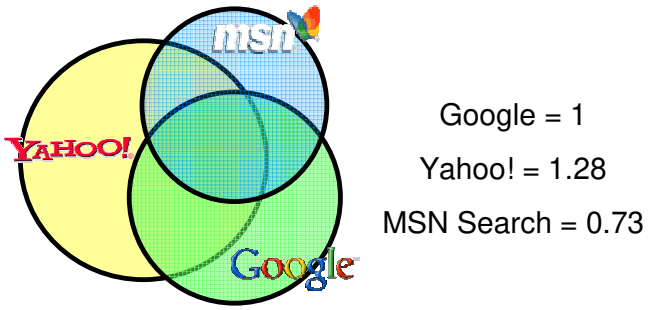

Figure 14: Relative corpus size of Google, MSN Search, and Yahoo!.

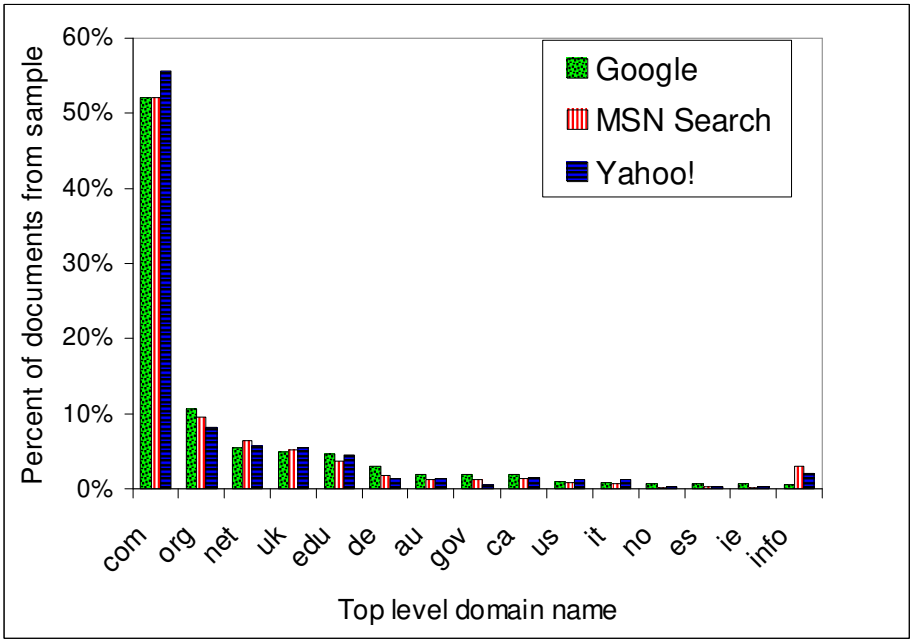

Figure 15: Top-level domain name distribution of pages in Google, MSN Search, and Yahoo! corpora.

9.5.5 Corpus freshness

We evaluated the freshness of the three corpora. First, we checked the percentage of pages indexed that are still valid. Figure 16 shows the percentage of dead pages (ones returning a 4xx HTTP return code) in the three corpora. Note that the Google corpus contains significantly more dead pages than Yahoo! and MSN Search.

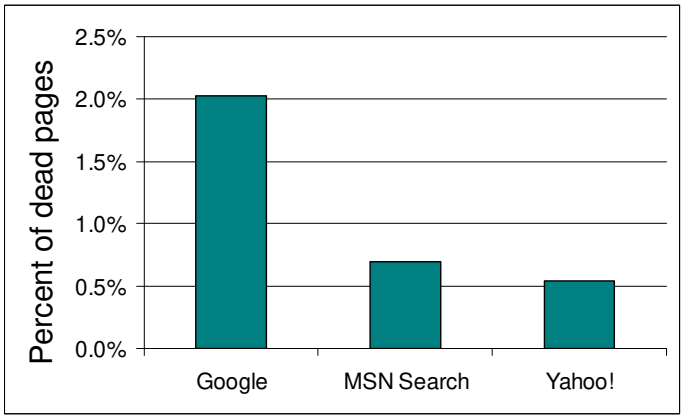

Figure 16: Percentage of inaccessible pages in the Google, MSN Search, and Yahoo! corpora.

Next, we tried to determine to what degree each of the three corpora reflects the current content of indexed web pages. To this end, we assumed that when the cached copy of the document is up-to-date, the index is up-to-date too and vice versa. We compared the two versions of the sampled document: (1) the cached document, and (2) the actual document fetched from the web. These measurements were performed on samples of HTML documents only, for which we were able to fetch both the cached copy and the document itself. Fetching of both document versions was performed simultaneously.

Figure 17 shows, for each value $0 \leq p \leq 100$, the fraction of samples, for which at most p percent of the lines in the cached version are different from the web version. The Yahoo! results look rather bad especially for low p values. After looking at the data, we found that some of the cached documents stored by Yahoo! were slightly pre-processed.

To provide a more objective measure of freshness, we measured text-only difference, which would ignore all formatting or other non-essential differences between the two document versions. We compared the bag-of-words representations of the two versions of each sample document. Figure 18 shows, for each value $0 \leq p \leq 100$, the fraction of samples, for which at most p percent of the words in the cached version are different from the web version.

We can see that about 55% - 60% of the documents are "fresh" in all three search engines, while Google's freshness is the lowest and MSN's is the highest.

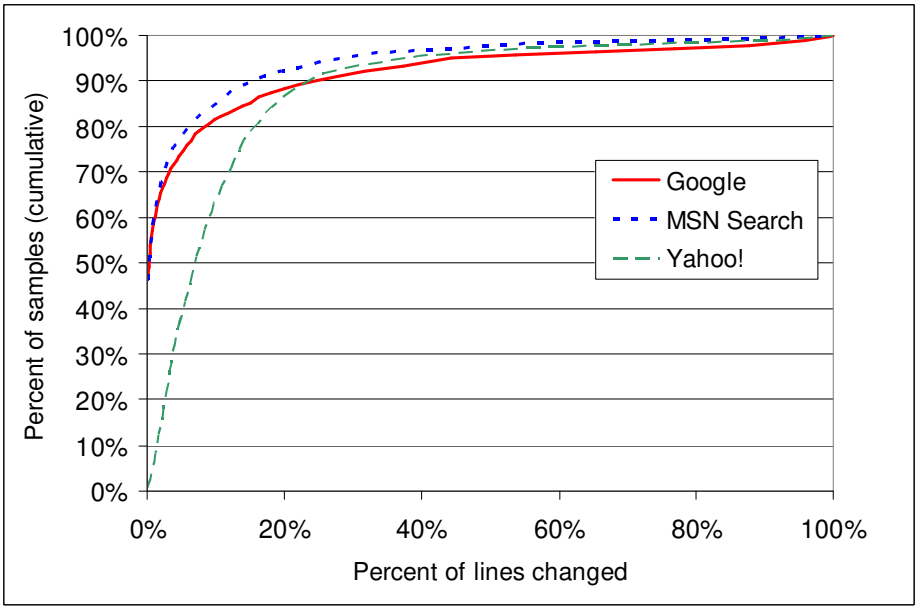

Figure 17: Raw HTML freshness of the Google, MSN Search, and Yahoo! corpora.

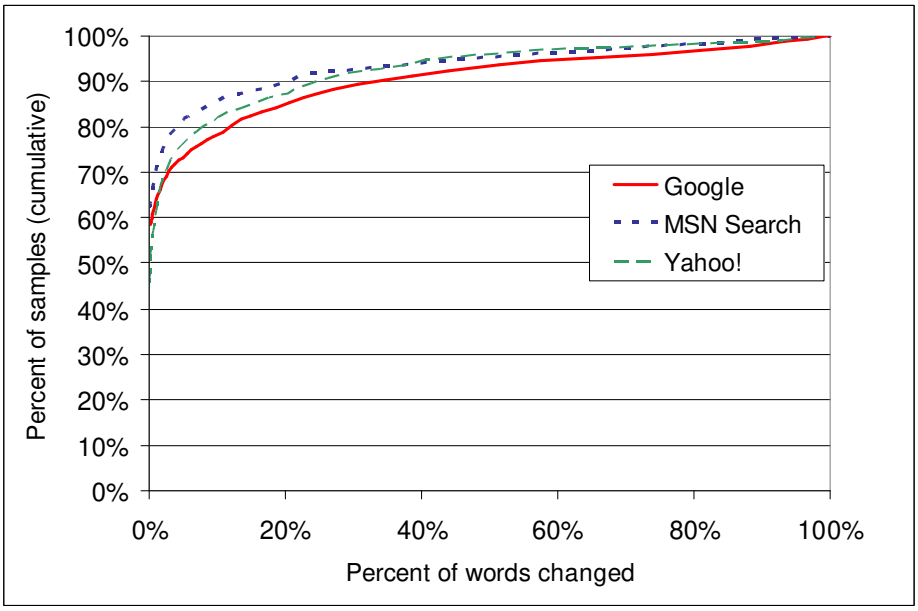

Figure 18: Text freshness of the Google, MSN Search, and Yahoo! corpora.

9.5.6 Percentage of dynamic pages

In this experiment we calculated the percentage of dynamic pages in the three corpora. We deem a page "dynamic" if its URL contains the characters "?" or "&" or if the URL ends with one of the following filename extensions: `.php, .php3, .asp, .cfm, .cgi, .pl, .jsp, .exe, .dll`. Figure 19 shows substantial differences among the search engines in terms of the percentage of dynamic pages in their corpora: Google has the highest percentage, while MSN has the lowest. This may indicate Google is doing a better job in crawling "deep web" dynamic content.

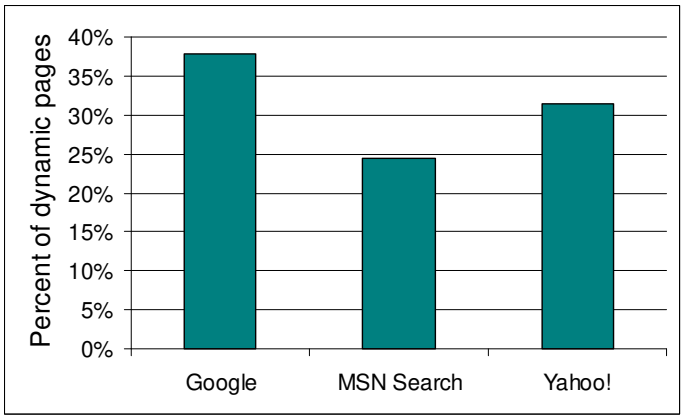

Figure 19: Percentage of dynamic pages in the Google, MSN Search, and Yahoo! corpora.

10 Conclusions

We presented two novel search engine samplers. The first, the pool-based sampler, uses a pre-prepared pool of queries to generate random queries. Random documents are then selected from the results of these queries. The sampler employs a Monte Carlo method, like rejection sampling, to guarantee that the distribution of the samples is close to the target distribution. We show that the sampler works, even if the sampling "weights", which are needed by the Monte Carlo method, are only approximate.

We provided full analysis of the sampler and identified the query pool parameters that impact its bias and performance. We then estimated these parameters on real data, consisting of the English pages from the ODP hierarchy, and showed that a pool of 5-term phrases achieves the best tradeoff between accuracy and performance.

Our second sampler runs a random walk on a graph defined over the indexed documents. Its primary advantage is that it does not need a pre-prepared query pool. This sampler employs a Markov Chain Monte Carlo method, like the Metropolis-Hastings algorithm or the Maximum Degree method, to guarantee that the random walk converges to the target distribution. Theoretical bounds on the convergence rate of the random walk are quite high, yet practical evidence suggests that the random walk produces only slightly biased samples much before reaching the theoretical bounds.

Our experimental results bare bias towards pages in the English language, since the query pool we

used consisted primarily of English phrase queries. We note that this bias is a limitation of our experimental setup and not of the techniques themselves. The bias could be eliminated by using a more comprehensive query pool.

Although we tested our samplers on web search engines only, we speculate that they may be applicable in a more general setting. For example, for sampling from databases, deep web sites, library records, medical data, etc.

As the query and fetch cost of our samplers are quite high, it remains as an open problem to design much more efficient search engine samplers.

References

[1] D. Aldous. On the Markov chain simulation method for uniform combinatorial distributed and simulated annealing. *Probability in the Engineering and Informational Sciences*, 1:33–46, 1987.

[2] A. Anagnostopoulos, A. Broder, and D. Carmel. Sampling search-engine results. In *Proceedings of the 14th International World Wide Web Conference (WWW)*, pages 245–256, 2005.

[3] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about Web pages via random walks. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*, pages 535–544, 2000.

[4] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, pages 367–376, 2006.

[5] J. Battelle. John Battelle's searchblog. `http://battellemedia.com/archives/001889.php`, 2005.

[6] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of the 7th International World Wide Web Conference (WWW7)*, pages 379–388, 1998.

[7] N. Blachman. Google guide. `http://www.googleguide.com`.

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

[9] E. T. Bradlow and D. C. Schmittlein. The little engines that could: Modeling the performance of World Wide Web search engines. *Marketing Science*, 19:43–62, 2000.

[10] A. Broder, M. Fontoura, V. Josifovski, R. Kumar, R. Motwani, S. Nabar, R. Panigrahy, A. Tomkins, and Y. Xu. Estimating corpus size via queries. In *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management*, 2006. To appear.

[11] F. Can, R. Nuray, and A. B. Sevdik. Automatic performance evaluation of Web search engines. *Information Processing and Management*, 40:495–514, 2004.

[12] M. Cheney and M. Perry. A comparison of the size of the Yahoo! and Google indices. Available at `http://vburton.ncsa.uiuc.edu/indexsize.html`, 2005.

[13] B. D. Davison. The potential of the metasearch engine. In *Proceedings of the Annual Meeting of the American Society for Information Science and Technology (ASIST)*, volume 41, pages 393–402, 2004.

[14] P. Diaconis and L. Saloff-Coste. What do we know about the Metropolis algorithm? *J. of Computer and System Sciences*, 57:20–36, 1998.

[15] dmoz. The open directory project. `http://dmoz.org`.

[16] A. Dobra and S. E. Fienberg. How large is the World Wide Web? *Web Dynamics*, pages 23–44, 2004.

[17] D. Gillman. A Chernoff bound for random walks on expander graphs. *SIAM J. on Computing*, 27(4):1203–1220, 1998.

[18] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[19] Google Inc. Google. `http://www.google.com`.

[20] M. Gordon and P. Pathak. Finding information on the World Wide Web: the retrieval effectiveness of search engines. *Information Processing and Management*, 35(2):141–180, 1999.

[21] A. Gulli and A. Signorini. The indexable Web is more than 11.5 billion pages. In *Proceedings of the 14th international conference on World Wide Web (WWW) - Special interest tracks and posters*, pages 902–903, 2005.

[22] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[23] D. Hawking, N. Craswel, P. Bailey, and K. Griffiths. Measuring search engine quality. *Information Retrieval*, 4(1):33–59, 2001.

[24] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. Measuring index quality using random walks on the Web. In *Proceedings of the 8th International World Wide Web Conference*, pages 213–225, May 1999.

[25] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform URL sampling. In *Proceedings of the 9th International World Wide Web Conference*, pages 295–308, May 2000.

[26] T. C. Hesterberg. *Advances in Importance Sampling*. PhD thesis, Stanford University, 1988.

[27] N. Kahale. Large deviation for Markov chains. *Combinatorics, Probability and Computing*, 6:465–474, 1997.

[28] S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science*, 5360(280):98, 1998.

[29] S. Lawrence and C. L. Giles. Accessibility of information on the Web. *Nature*, 400:107–109, 1999.

[30] J. S. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6:113–119, 1996.

[31] J. S. Liu. Monte Carlo Strategies in Scientific Computing. Springer, 2001.

[32] A. Marshal. The use of multi-stage sampling schemes in Monte Carlo computations. In M. Meyer, editor, *Symposium on Monte Carlo Methods*, volume 21, pages 123–140, New York, 1956. Wiley.

[33] T. Mayer. Our blog is growing up - and so has our index. Available at `http://www.ysearchblog.com/archives/000172.html`, 2005.

[34] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. of Chemical Physics*, 21:1087–1091, 1953.

[35] Microsoft Corporation. MSN Search. `search.msn.com`.

[36] G. Price. More on the total database size battle and Googlewhacking with Yahoo. Available at `http://blog.searchenginewatch.com/blog/050811-231448`, 2005.

[37] P. Rusmevichientong, D. Pennock, S. Lawrence, and C. L. Giles. Methods for sampling pages uniformly from the World Wide Web. In *Proceedings of AAAI Fall Symposium on Using Uncertainty within Computation*, 2001.

[38] D. Siegmund. *Sequential Analysis - Tests and Confidence Intervals*. Springer-Verlag, 1985.

[39] A. J. Sinclair and M. R. Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82:93–133, 1989.

[40] J. Véronis. Yahoo: Missing pages? (2). `http://aixtal.blogspot.com/2005/08/yahoo-missing-pages-2.html`, 2005.

[41] J. von Neumann. Various techniques used in connection with random digits. In *John von Neumann, Collected Works*, volume V. Oxford, 1963.

[42] Yahoo! Inc. Yahoo! `http://www.yahoo.com`.

A Approximate Monte Carlo methods – Proofs

A.1 Approximate rejection sampling

Theorem 6 (restated) *The sampling distribution of the approximate rejection sampling procedure is:*

$$\pi'(x) \;=\; \pi(x)\;\frac{\frac{p(x)}{q(x)}}{\mathbb{E}_\pi\left(\frac{p(X)}{q(X)}\right)}.$$

The expected number of samples from p needed to generate each sample from π' is:

$$\frac{C Z_{\hat{q}}}{Z_{\hat{\pi}} \ \mathbb{E}_{\pi}\left(\frac{p(X)}{q(X)}\right)}.$$

Proof. We start by finding a closed form expression for π'. Let X denote a random sample from p. Let Z be a 0-1 random variable, which is 1 with probability $r'_{\mathrm{RS}}(\mathrm{X})$ and 0 otherwise. Z corresponds to the outcome of the coin toss in the acceptance-rejection procedure of approximate rejection sampling. The procedure returns X as a sample if and only if $\mathrm{Z} = 1$. Therefore, π' is the distribution of X conditioned on the event "$\mathrm{Z} = 1$". In other words, for all $\mathrm{x} \in \mathrm{supp}(\pi)$,

$$\pi'(\mathrm{x}) = \Pr(\mathrm{X} = \mathrm{x} | \mathrm{Z} = 1).$$

By Bayes rule,

$$\Pr(\mathrm{X} = \mathrm{x} | \mathrm{Z} = 1) \ = \ \frac{\Pr(\mathrm{Z} = 1 | \mathrm{X} = \mathrm{x}) \cdot \Pr(\mathrm{X} = \mathrm{x})}{\Pr(\mathrm{Z} = 1)} \ = \ \frac{r'_{\mathrm{RS}}(\mathrm{x}) \ p(\mathrm{x})}{\Pr(\mathrm{Z} = 1)}.$$

Expanding over all possibilities for X, we have:

$$\Pr(\mathrm{Z} = 1) \ = \ \sum_{\mathrm{y} \in \mathrm{supp}(\pi)} \Pr(\mathrm{Z} = 1 | \mathrm{X} = \mathrm{y}) \cdot \Pr(\mathrm{X} = \mathrm{y}) \ = \ \sum_{\mathrm{y} \in \mathrm{supp}(\pi)} r'_{\mathrm{RS}}(\mathrm{y}) \ p(\mathrm{y}).$$

Hence,

$$\Pr(\mathrm{X} = \mathrm{x} | \mathrm{Z} = 1) \ = \ \frac{r'_{\mathrm{RS}}(\mathrm{x}) \ p(\mathrm{x})}{\sum_{\mathrm{y} \in \mathrm{supp}(\pi)} r'_{\mathrm{RS}}(\mathrm{y}) \ p(\mathrm{y})}.$$

Recalling the definition of r'_{RS}, we have:

$$\begin{aligned}
\pi'(\mathrm{x}) \ &= \ \Pr(\mathrm{X} = \mathrm{x} | \mathrm{Z} = 1) \ = \ \frac{\frac{\hat{\pi}(\mathrm{x})}{C\hat{q}(\mathrm{x})} \ p(\mathrm{x})}{\sum_{\mathrm{y} \in \mathrm{supp}(\pi)} \frac{\hat{\pi}(\mathrm{y})}{C\hat{q}(\mathrm{y})} \ p(\mathrm{y})} \ = \ \frac{\frac{\pi(\mathrm{x}) Z_{\hat{\pi}}}{Cq(\mathrm{x}) Z_{\hat{q}}} \ p(\mathrm{x})}{\sum_{\mathrm{y} \in \mathrm{supp}(\pi)} \frac{\pi(\mathrm{y}) Z_{\hat{\pi}}}{Cq(\mathrm{y}) Z_{\hat{q}}} \ p(\mathrm{y})} \\
&= \ \frac{\pi(\mathrm{x}) \frac{p(\mathrm{x})}{q(\mathrm{x})}}{\sum_{\mathrm{y} \in \mathrm{supp}(\pi)} \pi(\mathrm{y}) \frac{p(\mathrm{y})}{q(\mathrm{y})}} \ = \ \pi(\mathrm{x}) \ \frac{\frac{p(\mathrm{x})}{q(\mathrm{x})}}{\mathbb{E}_{\pi}\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}.
\end{aligned}$$

We now turn to calculating the cost of the approximate rejection sampling procedure. The procedure generates samples from p until the acceptance-rejection procedure accepts. The probability of acceptance is $\Pr(\mathrm{Z} = 1)$, and thus the number of samples generated from p is a geometric random variable whose probability of success is $\Pr(\mathrm{Z} = 1)$. Its expectation is $1/\Pr(\mathrm{Z} = 1)$. Let us then calculate this probability:

$$\begin{aligned}
\Pr(\mathrm{Z} = 1) \ &= \ \sum_{\mathrm{y} \in \mathrm{supp}(\pi)} r'_{\mathrm{RS}}(\mathrm{y}) \ p(\mathrm{y}) \ = \ \sum_{\mathrm{y} \in \mathrm{supp}(\pi)} \frac{\pi(\mathrm{y}) Z_{\hat{\pi}}}{Cq(\mathrm{y}) Z_{\hat{q}}} \ p(\mathrm{y}) \\
&= \ \frac{Z_{\hat{\pi}}}{C Z_{\hat{q}}} \cdot \sum_{\mathrm{y} \in \mathrm{supp}(\pi)} \pi(\mathrm{y}) \frac{p(\mathrm{y})}{q(\mathrm{y})} \\
&= \ \frac{Z_{\hat{\pi}}}{C Z_{\hat{q}}} \cdot \mathbb{E}_{\pi}\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right).
\end{aligned}$$

□

Proposition 7 (restated)

$$||\pi' - \pi|| = \frac{1}{2}\,\mathrm{ndev}_\pi\left(\frac{p(X)}{q(X)}\right).$$

Proof.

$$\begin{aligned}
||\pi' - \pi|| &= \frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\pi)}\left|\pi'(\mathrm{x}) - \pi(\mathrm{x})\right| = \frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\pi)}\left|\pi(\mathrm{x})\ \frac{\frac{p(\mathrm{x})}{q(\mathrm{x})}}{\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)} - \pi(\mathrm{x})\right| \\
&= \frac{1}{2}\cdot\frac{1}{\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\cdot\sum_{\mathrm{x}\in\mathrm{supp}(\pi)}\pi(\mathrm{x})\ \left|\frac{p(\mathrm{x})}{q(\mathrm{x})} - \mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)\right| \\
&= \frac{1}{2}\cdot\frac{1}{\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\cdot\mathrm{dev}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right) \\
&= \frac{1}{2}\,\mathrm{ndev}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right).
\end{aligned}$$

□

A.2 Approximate importance sampling

Theorem 8 (restated) *Let*

$$\hat{\mu}' = \frac{\hat{\mu}'_1}{\hat{\mu}'_2} = \frac{\frac{1}{n}\sum_{i=1}^n f(X_i)w'(X_i)}{\frac{1}{n}\sum_{i=1}^n w'(X_i)}$$

be the estimator produced by the approximate importance sampling procedure for the parameter $E_\pi(f(X))$. Then,

$$\frac{\mathbb{E}_p(\hat{\mu}'_1)}{\mathbb{E}_p(\hat{\mu}'_2)} = \mathbb{E}_\pi(f(X)) + \frac{\mathrm{cov}_\pi\left(f(X), \frac{p(X)}{q(X)}\right)}{E_\pi\left(\frac{p(X)}{q(X)}\right)}.$$

Proof. Since $\mathrm{X}_1, \ldots, \mathrm{X}_n$ are i.i.d. random variables, it suffices to analyze the expectations of $f(\mathrm{X})w'(\mathrm{X})$ and $w'(\mathrm{X})$, where $\mathrm{X} \sim p$.

$$\begin{aligned}
\mathbb{E}_p(f(\mathrm{X})w'(\mathrm{X})) &= \sum_{\mathrm{x}\in\mathcal{U}} p(\mathrm{x})\ f(\mathrm{x})\ \frac{\hat{\pi}(\mathrm{x})}{\hat{q}(\mathrm{x})} \\
&= \frac{Z_{\hat{\pi}}}{Z_{\hat{q}}}\sum_{\mathrm{x}\in\mathcal{U}}\pi(\mathrm{x})\ f(\mathrm{X})\ \frac{p(\mathrm{x})}{q(\mathrm{x})} \\
&= \frac{Z_{\hat{\pi}}}{Z_{\hat{q}}}\ \mathbb{E}_\pi\left(f(\mathrm{X})\ \frac{p(\mathrm{X})}{q(\mathrm{X})}\right).
\end{aligned}$$

$$\begin{aligned}\mathbb{E}_p(w'(\mathrm{X})) &= \sum_{\mathrm{x}\in\mathcal{U}} p(\mathrm{x})\ \frac{\hat{\pi}(\mathrm{x})}{\hat{q}(\mathrm{x})} \;=\; \frac{Z_{\hat{\pi}}}{Z_{\hat{q}}} \sum_{\mathrm{x}\in\mathcal{U}} \pi(\mathrm{x})\ \frac{p(\mathrm{x})}{q(\mathrm{x})} \\ &= \frac{Z_{\hat{\pi}}}{Z_{\hat{q}}}\ \mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right).\end{aligned}$$

Therefore,

$$\begin{aligned}\frac{\mathbb{E}_p(\hat{\mu}'_1)}{\mathbb{E}_p(\hat{\mu}'_2)} &= \frac{\mathbb{E}_p(f(\mathrm{X})w'(\mathrm{X}))}{\mathbb{E}_p(w'(\mathrm{X}))} \;=\; \frac{\frac{Z_{\hat{\pi}}}{Z_{\hat{q}}}\cdot\mathbb{E}_\pi\left(f(\mathrm{X})\ \frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}{\frac{Z_{\hat{\pi}}}{Z_{\hat{q}}}\cdot\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)} \\ &= \frac{\mathrm{cov}_\pi\left(f(\mathrm{X}),\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)+\mathbb{E}_\pi(f(\mathrm{X}))\ \mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}{\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)} \\ &= \mathbb{E}_\pi(f(\mathrm{X})) + \frac{\mathrm{cov}_\pi\left(f(\mathrm{X}),\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}{\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}.\end{aligned}$$

□

A.3 Approximate Metropolis-Hastings

Theorem 9 (restated) *Let P'_{MH} be the transition matrix of the approximate Metropolis-Hastings algorithm. Then, P'_{MH} forms an ergodic Markov chain and its unique limit distribution is π'.*

Proof. The transition matrix of approximate MH is:

$$P'_{\mathrm{MH}}(\mathrm{x},\mathrm{y}) \;=\; \begin{cases} P(\mathrm{x},\mathrm{y})\ r'_{\mathrm{MH}}(\mathrm{x},\mathrm{y}), & \text{if } \mathrm{x}\neq\mathrm{y}, \\ P(\mathrm{x},\mathrm{x})\ r'_{\mathrm{MH}}(\mathrm{x},\mathrm{x}) + 1 - \sum_{z\in\mathcal{U}} P(\mathrm{x},\mathrm{z})\ r'_{\mathrm{MH}}(\mathrm{x},\mathrm{z}), & \text{if } \mathrm{x}=\mathrm{y}. \end{cases}$$

Recall that:

$$r'_{\mathrm{MH}}(\mathrm{x},\mathrm{y}) \;=\; \min\left\{\frac{\pi(\mathrm{y})\ q(\mathrm{x})}{\pi(\mathrm{x})\ q(\mathrm{y})},\ 1\right\}.$$

Since $\pi(\mathrm{x})>0$ and $q(\mathrm{x})>0$ for all $\mathrm{x}\in\mathcal{U}$, then $r'_{\mathrm{MH}}(\mathrm{x},\mathrm{y})>0$ for all $\mathrm{x},\mathrm{y}\in\mathcal{U}$. It follows that whenever $P(\mathrm{x},\mathrm{y})>0$, then also $P'_{\mathrm{MH}}(\mathrm{x},\mathrm{y})>0$. This implies that the Markov chain graph $G_{P'_{\mathrm{MH}}}$ corresponding to P'_{MH} contains all the edges of the Markov chain graph G_P corresponding to P. Since P is ergodic, we know that G_P is strongly connected and aperiodic. As strong connectedness and aperiodicity are invariant under addition of edges, it follows that also $G_{P'_{\mathrm{MH}}}$ must be strongly connected and aperiodic, and thus P'_{MH} is ergodic.

As P'_{MH} is ergodic, the fundamental theorem of Markov chains tell us that it has a unique limit distribution η. Furthermore, η is the unique stationary distribution of P'_{MH}. We use this fact to calculate a closed form expression for η. It will then be evident that $\eta=\pi'$.

Let R be the following $|\mathcal{U}| \times |\mathcal{U}|$ matrix:

$$R(\mathrm{x},\mathrm{y}) = P(\mathrm{x},\mathrm{y})\ r'_{\text{MH}}(\mathrm{x},\mathrm{y}).$$

Let D be the following $|\mathcal{U}| \times |\mathcal{U}|$ diagonal matrix:

$$D(\mathrm{x},\mathrm{y}) = \begin{cases} 0, & \text{if } \mathrm{x} \neq \mathrm{y}, \\ \sum_{\mathrm{z}\in\mathcal{U}} R(\mathrm{x},\mathrm{z}), & \text{if } \mathrm{x} = \mathrm{y}. \end{cases}$$

Let I be the $|\mathcal{U}| \times |\mathcal{U}|$ identity matrix. Using R, D, and I we can express P'_{MH} as follows:

$$P'_{\text{MH}} \ = \ R + I - D.$$

Therefore,

$$\pi' P'_{\text{MH}} \ = \ \pi' R + \pi' - \pi' D.$$

Thus, in order to show that π' is a stationary distribution of P'_{MH}, it suffices to prove that

$$\pi' R = \pi' D.$$

For any given $\mathrm{x}, \mathrm{y} \in \mathcal{U}$, there are three possible situations, depending on whether $\frac{\pi(\mathrm{y})\ q(\mathrm{x})}{\pi(\mathrm{x})\ q(\mathrm{y})}$ equals 1, is greater than 1, or is less than 1. We define three predicates ϕ_1, ϕ_2, ϕ_3 specifying these situations:

$$\phi_1(\mathrm{x},\mathrm{y}) \ = \ \begin{cases} 1, & \text{if } \frac{\pi(\mathrm{y})\ q(\mathrm{x})}{\pi(\mathrm{x})\ q(\mathrm{y})} = 1, \\ 0, & \text{otherwise.} \end{cases}$$

$$\phi_2(\mathrm{x},\mathrm{y}) \ = \ \begin{cases} 1, & \text{if } \frac{\pi(\mathrm{y})\ q(\mathrm{x})}{\pi(\mathrm{x})\ q(\mathrm{y})} > 1, \\ 0, & \text{otherwise.} \end{cases}$$

$$\phi_3(\mathrm{x},\mathrm{y}) \ = \ \begin{cases} 1, & \text{if } \frac{\pi(\mathrm{y})\ q(\mathrm{x})}{\pi(\mathrm{x})\ q(\mathrm{y})} < 1, \\ 0, & \text{otherwise.} \end{cases}$$

It is easy to verify that for every $\mathrm{x}, \mathrm{y} \in \mathcal{U}$, the following hold:

1. $\phi_1(\mathrm{x},\mathrm{y}) + \phi_2(\mathrm{x},\mathrm{y}) + \phi_3(\mathrm{x},\mathrm{y}) = 1$.
2. $\phi_1(\mathrm{x},\mathrm{y}) = \phi_1(\mathrm{y},\mathrm{x})$.
3. $\phi_2(\mathrm{x},\mathrm{y}) = \phi_3(\mathrm{y},\mathrm{x})$ and $\phi_3(\mathrm{x},\mathrm{y}) = \phi_2(\mathrm{y},\mathrm{x})$.
4. If $\phi_1(\mathrm{x},\mathrm{y}) = 1$ or $\phi_2(\mathrm{x},\mathrm{y}) = 1$, then $r'_{\text{MH}}(\mathrm{x},\mathrm{y}) = 1$.
5. If $\phi_1(\mathrm{x},\mathrm{y}) = 1$ or $\phi_3(\mathrm{x},\mathrm{y}) = 1$, then $r'_{\text{MH}}(\mathrm{x},\mathrm{y}) = \frac{\pi(\mathrm{y})\ q(\mathrm{x})}{\pi(\mathrm{x})\ q(\mathrm{y})}$.

We now use these predicates to break R and D into pieces. For each $i = 1, 2, 3$, let R_i be the following matrix:

$$R_i(\mathrm{x},\mathrm{y}) = R(\mathrm{x},\mathrm{y})\ \phi_i(\mathrm{x},\mathrm{y}).$$

Clearly,

$$R = R_1 + R_2 + R_3.$$

For each $i = 1, 2, 3$, let D_i be a diagonal matrix whose diagonal elements are the following:

$$D_i(\mathrm{x}, \mathrm{x}) = \sum_{\mathrm{z} \in \mathcal{U}} R_i(\mathrm{x}, \mathrm{z}).$$

It can be easily verified that since $R = R_1 + R_2 + R_3$, then also:

$$D = D_1 + D_2 + D_3.$$

In order to prove that $\pi' R = \pi' D$, we will prove that:

$$\pi' R_1 = \pi' D_1, \quad \pi' R_2 = \pi' D_3, \quad \pi' R_3 = \pi' D_2. \tag{1}$$

Before we prove these equalities, we observe the following characterizations of R_1, R_2, R_3. For $i = 1, 2$,

$$R_i(\mathrm{x}, \mathrm{y}) \;=\; P(\mathrm{x}, \mathrm{y})\; r'_{\mathrm{MH}}(\mathrm{x}, \mathrm{y})\; \phi_i(\mathrm{x}, \mathrm{y}) \;=\; P(\mathrm{x}, \mathrm{y})\; \phi_i(\mathrm{x}, \mathrm{y}),$$

while for $i = 1, 3$,

$$R_i(\mathrm{x}, \mathrm{y}) \;=\; P(\mathrm{x}, \mathrm{y})\; r'_{\mathrm{MH}}(\mathrm{x}, \mathrm{y})\; \phi_i(\mathrm{x}, \mathrm{y}) \;=\; P(\mathrm{x}, \mathrm{y})\; \frac{\pi(\mathrm{y})\; q(\mathrm{x})}{\pi(\mathrm{x})\; q(\mathrm{y})}\; \phi_i(\mathrm{x}, \mathrm{y}).$$

(Note that R_1 has two equivalent characterizations.)

We can now prove the equalities in Equation (1). Let $i \in \{1, 2\}$. Using the characterization of R_1, R_2 and the definition of π', we have for every $\mathrm{x} \in \mathcal{U}$,

$$\begin{aligned}
(\pi' R_i)(\mathrm{x}) \;&=\; \sum_{\mathrm{y} \in \mathcal{U}} \pi'(\mathrm{y})\; R_i(\mathrm{y}, \mathrm{x}) \\
&=\; \sum_{\mathrm{y} \in \mathcal{U}} \pi(\mathrm{y})\; \frac{\frac{p(\mathrm{y})}{q(\mathrm{y})}}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{y}, \mathrm{x})\; \phi_i(\mathrm{y}, \mathrm{x}).
\end{aligned}$$

As P is a reversible Markov chain, $P(\mathrm{y}, \mathrm{x}) = P(\mathrm{x}, \mathrm{y}) \frac{p(\mathrm{x})}{p(\mathrm{y})}$. Furthermore, recall that $\phi_i(\mathrm{y}, \mathrm{x}) = \phi_j(\mathrm{x}, \mathrm{y})$, where if $i = 1$, then $j = 1$, and if $i = 2$, then $j = 3$. Therefore,

$$\begin{aligned}
(\pi' R_i)(\mathrm{x}) \;&=\; \sum_{\mathrm{y} \in \mathcal{U}} \pi(\mathrm{y})\; \frac{\frac{p(\mathrm{y})}{q(\mathrm{y})}}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{x}, \mathrm{y})\; \frac{p(\mathrm{x})}{p(\mathrm{y})}\; \phi_j(\mathrm{x}, \mathrm{y}) \\
&=\; \sum_{\mathrm{y} \in \mathcal{U}} \frac{p(\mathrm{x})}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{x}, \mathrm{y})\; \frac{\pi(\mathrm{y})}{q(\mathrm{y})}\; \phi_j(\mathrm{x}, \mathrm{y}).
\end{aligned}$$

By multiplying and dividing each term by $\frac{q(\mathrm{x})}{\pi(\mathrm{x})}$, we obtain:

$$(\pi' R_i)(\mathrm{x}) \;=\; \sum_{\mathrm{y} \in \mathcal{U}} \pi(\mathrm{x})\; \frac{\frac{p(\mathrm{x})}{q(\mathrm{x})}}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{x}, \mathrm{y})\; \frac{\pi(\mathrm{y})\; q(\mathrm{x})}{\pi(\mathrm{x})\; q(\mathrm{y})}\; \phi_j(\mathrm{x}, \mathrm{y}).$$

Using the definition of π' and the characterization for R_1, R_3, we have:

$$(\pi' R_i)(\mathrm{x}) \;=\; \pi'(\mathrm{x}) \sum_{\mathrm{y}\in\mathcal{U}} R_j(\mathrm{x},\mathrm{y}) \;=\; \pi'(\mathrm{x})\; D_j(\mathrm{x},\mathrm{x}) \;=\; (\pi' D_j)(\mathrm{x}).$$

This proves that $\pi' R_1 = \pi' D_1$ and $\pi' R_2 = \pi' D_3$. We are left to prove that $\pi' R_3 = \pi' D_2$. Using the characterization of R_3 and the definition of π', we have for every $\mathrm{x} \in \mathcal{U}$,

$$\begin{aligned}
(\pi' R_3)(\mathrm{x}) &= \sum_{\mathrm{y}\in\mathcal{U}} \pi'(\mathrm{y})\; R_3(\mathrm{y},\mathrm{x}) \\
&= \sum_{\mathrm{y}\in\mathcal{U}} \pi(\mathrm{y})\; \frac{\frac{p(\mathrm{y})}{q(\mathrm{y})}}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{y},\mathrm{x})\; \frac{\pi(\mathrm{x})\; q(\mathrm{y})}{\pi(\mathrm{y})\; q(\mathrm{x})}\; \phi_3(\mathrm{y},\mathrm{x}) \\
&= \sum_{\mathrm{y}\in\mathcal{U}} \pi(\mathrm{x})\; \frac{\frac{p(\mathrm{y})}{q(\mathrm{x})}}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{y},\mathrm{x})\; \phi_3(\mathrm{y},\mathrm{x}).
\end{aligned}$$

Using the reversibility of P and the fact $\phi_3(\mathrm{y},\mathrm{x}) = \phi_2(\mathrm{x},\mathrm{y})$, we have:

$$\begin{aligned}
(\pi' R_3)(\mathrm{x}) &= \sum_{\mathrm{y}\in\mathcal{U}} \pi(\mathrm{x})\; \frac{\frac{p(\mathrm{y})}{q(\mathrm{x})}}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{x},\mathrm{y})\; \frac{p(\mathrm{x})}{p(\mathrm{y})}\phi_2(\mathrm{x},\mathrm{y}) \\
&= \sum_{\mathrm{y}\in\mathcal{U}} \pi(\mathrm{x})\; \frac{\frac{p(\mathrm{x})}{q(\mathrm{x})}}{E_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)}\; P(\mathrm{x},\mathrm{y})\phi_2(\mathrm{x},\mathrm{y}).
\end{aligned}$$

By the definition of π and the characterization of R_2, we have:

$$(\pi' R_3)(\mathrm{x}) \;=\; \pi'(\mathrm{x}) \sum_{\mathrm{y}\in\mathcal{U}} P(\mathrm{x},\mathrm{y})\; \phi_2(\mathrm{x},\mathrm{y}) \;=\; \pi'(\mathrm{x}) \sum_{\mathrm{y}\in\mathcal{U}} R_2(\mathrm{x},\mathrm{y}) \;=\; \pi'(\mathrm{x}) D_2(\mathrm{x},\mathrm{x}) \;=\; (\pi' D_2)(\mathrm{x}).$$

This shows that $\pi' R_3 = \pi' D_2$. □

A.4 Approximate Maximum Degree

Theorem 10 (restated) *Let P'_{MD} be the transition matrix of the approximate Maximum Degree procedure. Then, P'_{MD} forms an ergodic Markov chain. The unique limit distribution of P'_{MD} is π'.*

Proof. The transition matrix of approximate MD is:

$$P'_{\mathrm{MD}}(\mathrm{x},\mathrm{y}) \;=\; \begin{cases} P(\mathrm{x},\mathrm{y})\; r'_{\mathrm{MD}}(\mathrm{x}), & \text{if } \mathrm{x} \neq \mathrm{y}, \\ P(\mathrm{x},\mathrm{x})\; r'_{\mathrm{MD}}(\mathrm{x}) + 1 - r'_{\mathrm{MD}}(\mathrm{x}), & \text{if } \mathrm{x} = \mathrm{y}. \end{cases}$$

Since $\hat{q}(\mathrm{x}) > 0$ for all $\mathrm{x} \in \mathcal{U}$, it can be seen from the above expression that for all $\mathrm{x} \neq \mathrm{y} \in \mathcal{U}$, $P'_{\mathrm{MD}}(\mathrm{x},\mathrm{y}) > 0$ if and only if $P(\mathrm{x},\mathrm{y}) > 0$. Furthermore, if $P'_{\mathrm{MD}}(\mathrm{x},\mathrm{x}) = 0$, then also $P(\mathrm{x},\mathrm{x}) = 0$ (the converse is not necessarily true). We conclude that the Markov chain graph $G_{P'_{\mathrm{MD}}}$ corresponding to P'_{MD} contains all the edges of the Markov chain graph G_P corresponding to P (the only possible

extra edges in $G_{P'_{\text{MD}}}$ are self loops). As P is ergodic, and ergodicity is invariant under addition of edges to the Markov chain graph, we conclude that also P'_{MD} is ergodic.

We next prove that π' is a stationary distribution of P'_{MD}. This would imply that π' is also the unique limit distribution of P'_{MD}.

Let Q be the following $|\mathcal{U}| \times |\mathcal{U}|$ diagonal matrix:

$$Q(\mathrm{x},\mathrm{y}) = \begin{cases} 0, & \text{if } \mathrm{x} \neq \mathrm{y}, \\ \frac{\hat{q}(\mathrm{x})}{C\ \hat{\pi}(\mathrm{x})}, & \text{if } \mathrm{x} = \mathrm{y}. \end{cases}$$

Let I be the $|\mathcal{U}| \times |\mathcal{U}|$ identity matrix. Using Q and I we can express P'_{MD} in terms of P:

$$P'_{\text{MD}} \ = \ QP + I - Q.$$

Since η is a stationary distribution of P'_{MD}, we have: $\eta P'_{\text{MD}} = \eta$. Therefore,

$$\eta(QP + I - Q) = \eta.$$

Hence,

$$\eta QP = \eta Q.$$

Let $\eta' = \frac{\eta Q}{||\eta Q||_1}$ be the normalized form of ηQ. Note that also η' is a fixed point of P: $\eta' P = \eta'$. Furthermore, η' is a non-negative vector, whose entries sum to 1. Therefore, η' is in fact a stationary distribution of P. However, as P is ergodic, p is the unique stationary distribution of P. We thus conclude:

$$\frac{\eta Q}{||\eta Q||_1} \ = \ \eta' \ = \ p.$$

Hence,

$$\frac{\eta}{||\eta Q||_1} \ = \ pQ^{-1}. \tag{2}$$

Thus, η and pQ^{-1} are equal, up to a scalar factor. Recall that η is a probability distribution, and thus $||\eta||_1 = 1$. Therefore, we must have:

$$\eta \ = \ \frac{pQ^{-1}}{||pQ^{-1}||_1}.$$

Let us further expand this formula. For each $\mathrm{x} \in \mathcal{U}$, we have:

$$(pQ^{-1})(\mathrm{x}) \ = \ \sum_{\mathrm{y} \in \mathcal{U}} p(\mathrm{y})\ Q^{-1}(\mathrm{y},\mathrm{x}) \ = \ p(\mathrm{x})\ \frac{C\ \hat{\pi}(\mathrm{x})}{\hat{q}(\mathrm{x})} \ = \ \pi(\mathrm{x})\ \frac{p(\mathrm{x})}{q(\mathrm{x})} \cdot \frac{C\ Z_{\hat{\pi}}}{Z_{\hat{q}}}.$$

Therefore,

$$\begin{aligned} ||pQ^{-1}||_1 \ &= \ \sum_{\mathrm{x} \in \mathcal{U}} |(pQ^{-1})(\mathrm{x})| \ = \ \sum_{\mathrm{x} \in \mathcal{U}} \pi(\mathrm{x})\ \frac{p(\mathrm{x})}{q(\mathrm{x})} \cdot \frac{C\ Z_{\hat{\pi}}}{Z_{\hat{q}}} \\ &= \ \frac{C\ Z_{\hat{\pi}}}{Z_{\hat{q}}} \ \sum_{\mathrm{x} \in \mathcal{U}} \pi(\mathrm{x})\ \frac{p(\mathrm{x})}{q(\mathrm{x})} \ = \ \frac{C\ Z_{\hat{\pi}}}{Z_{\hat{q}}} \ \mathbb{E}_{\pi}\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right). \end{aligned}$$

We conclude that for all $\mathrm{x} \in \mathcal{U}$,

$$\eta(\mathrm{x}) \;=\; \frac{(pQ^{-1})(\mathrm{x})}{||pQ^{-1}||_1} \;=\; \pi(\mathrm{x})\,\frac{p(\mathrm{x})}{q(\mathrm{x})} \cdot \frac{C\ Z_{\hat{\pi}}}{Z_{\hat{q}}} \cdot \frac{Z_{\hat{q}}}{C\ Z_{\hat{\pi}}} \cdot \frac{1}{\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)} \;=\; \pi(\mathrm{x})\,\frac{\frac{p(\mathrm{x})}{q(\mathrm{x})}}{\mathbb{E}_\pi\left(\frac{p(\mathrm{X})}{q(\mathrm{X})}\right)} \;=\; \pi'(\mathrm{x}).$$

□

B Pool-based sampler – Proofs

B.1 Analysis of sampling recall and sampling bias

Theorem 18 (restated) *The sampling recall of the PB sampler is equal to:*

$$\mathrm{recall}_\pi(\mathcal{P}_+) = \pi(\mathcal{D}_{\mathcal{P}_+}).$$

The sampling bias of the PB sampler is at most:

$$\frac{1}{2}\,\mathrm{ndev}_{\pi_{\mathcal{P}_+}}(\mathrm{vdensity}_{\mathcal{P}}(X)),$$

where $\pi_{\mathcal{P}_+}$ is the restriction of π to $\mathcal{D}_{\mathcal{P}_+} \cap \mathrm{supp}(\pi)$.

Proof. The analysis of the recall of the PB sampler is identical to the recall analysis shown in the proof of Proposition 15. We therefore omit the details.

Let η be the sampling distribution of the PB sampler. Similarly to the proof of Proposition 15, it can be shown that:

$$\mathrm{supp}(\eta) = \mathcal{D}_{\mathcal{P}_+} \cap \mathrm{supp}(\pi).$$

Hence, the restriction of π to $\mathrm{supp}(\eta)$, i.e., $\pi_{\mathrm{supp}(\eta)}$, is exactly $\pi_{\mathcal{P}_+}$. The sampling bias of the PB sampler is therefore the distance $||\eta - \pi_{\mathcal{P}_+}||$.

We note that the scenario we face here is exactly the one captured by Theorem 6: we have a rejection sampling procedure whose trial distribution does not match the unnormalized trial weights. The target distribution in this case is $\hat{\pi}_{\mathcal{P}_+}$ and the trial distribution is $d_{\mathcal{P}_+}$. Let us denote the distribution induced by the trial weights by q. We next calculate a closed form expression for q.

The support of q is the same as the support of $d_{\mathcal{P}_+}$, because only documents that are sampled from $d_{\mathcal{P}_+}$ are assigned an unnormalized trial weight. Hence, $\mathrm{supp}(q) = \mathrm{supp}(d_{\mathcal{P}_+}) = D_{\mathcal{P}_+}$. For each $\mathrm{x} \in D_{\mathcal{P}_+}$ the unnormalized trial weight is $\deg_{\mathcal{P}}(\mathrm{x})$. Therefore, the normalization constant is: $Z_{\hat{q}} = \sum_{\mathrm{x} \in \mathcal{D}_{\mathcal{P}_+}} \deg_{\mathcal{P}}(\mathrm{x}) = \deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})$. Therefore, for every document $\mathrm{x} \in \mathcal{D}_{\mathcal{P}_+}$ we have:

$$q(\mathrm{x}) \;=\; \frac{\deg_{\mathcal{P}}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})}.$$

Applying Theorem 6, we bound the bias of the PB sampler as follows:

$$||\eta - \pi_{\mathcal{P}_+}|| \;\leq\; \frac{1}{2}\,\mathrm{ndev}_{\pi_{\mathcal{P}_+}}\left(\frac{d_{\mathcal{P}_+}(\mathrm{X})}{q(\mathrm{X})}\right).$$

For each $\mathrm{x} \in \mathcal{D}_{\mathcal{P}_+}$, we have:

$$\frac{d_{\mathcal{P}_+}(\mathrm{x})}{q(\mathrm{x})} \;=\; \frac{\frac{\deg_{\mathcal{P}_+}(\mathrm{x})}{\deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}}{\frac{\deg_{\mathcal{P}}(\mathrm{x})}{\deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})}} \;=\; \mathrm{vdensity}_{\mathcal{P}}(\mathrm{x})\,\frac{\deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})}{\deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}.$$

Therefore,

$$\mathrm{ndev}_{\pi_{\mathcal{P}_+}}\left(\frac{d_{\mathcal{P}_+}(\mathrm{X})}{q(\mathrm{X})}\right) \;=\; \mathrm{ndev}_{\pi_{\mathcal{P}_+}}\left(\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X})\cdot\frac{\deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})}{\deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}\right) \;=\; \mathrm{ndev}_{\pi_{\mathcal{P}_+}}\left(\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X})\right),$$

where the latter equality follows from the fact the normalized mean deviation is invariant under multiplication by a scalar. □

Theorem 19 (restated) *The sampling bias of the PB sampler is at most:*

$$\frac{\mathrm{ovprob}(w_{\mathcal{P}})}{1-\mathrm{ovprob}(w_{\mathcal{P}})}.$$

Proof. In order to prove the theorem, we prove that

$$\frac{1}{2}\cdot\mathrm{ndev}_{\pi_{\mathcal{P}_+}}(\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X})) \;\le\; \frac{\mathrm{ovprob}(w_{\mathcal{P}})}{1-\mathrm{ovprob}(w_{\mathcal{P}})}.$$

Let $\mu = \mathbb{E}_{\pi_{\mathcal{P}_+}}(\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X}))$. Then,

$$\frac{1}{2}\cdot\mathrm{ndev}_{\pi_{\mathcal{P}_+}}(\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X})) \;=\; \frac{\mathbb{E}_{\pi_{\mathcal{P}_+}}(|\,\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X})-\mu|)}{2\mu}.$$

For a document $\mathrm{x} \in \mathcal{D}_{\mathcal{P}_+}$, we define the *invalidity density* of x as:

$$\mathrm{invdensity}_{\mathcal{P}}(\mathrm{x}) \;=\; 1-\mathrm{vdensity}_{\mathcal{P}}(\mathrm{x}).$$

Then,

$$\frac{\mathbb{E}_{\pi_{\mathcal{P}_+}}(|\,\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X})-\mu|)}{2\mu} \;=\; \frac{\mathbb{E}_{\pi_{\mathcal{P}_+}}(|1-\mathrm{invdensity}_{\mathcal{P}}(\mathrm{X})-\mu|)}{2\mu}.$$

By the triangle inequality, for every $\mathrm{x} \in \mathcal{D}_{\mathcal{P}_+}$,

$$|1-\mathrm{invdensity}_{\mathcal{P}}(\mathrm{x})-\mu| \;\le\; 1-\mu+\mathrm{invdensity}_{\mathcal{P}}(\mathrm{x}) \;=\; 1-\mu+1-\mathrm{vdensity}_{\mathcal{P}}(\mathrm{x}).$$

Hence,

$$\begin{aligned}\frac{\mathbb{E}_{\pi_{\mathcal{P}_+}}(|1-\mathrm{invdensity}_{\mathcal{P}}(\mathrm{X})-\mu|)}{2\mu} &\le \frac{\mathbb{E}_{\pi_{\mathcal{P}_+}}(1-\mu+1-\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X}))}{2\mu}\\ &= \frac{1-\mu+1-\mathbb{E}_{\pi_{\mathcal{P}_+}}(\mathrm{vdensity}_{\mathcal{P}}(\mathrm{X}))}{2\mu}\\ &= \frac{1-\mu+1-\mu}{2\mu} \;=\; \frac{1-\mu}{\mu}.\end{aligned} \tag{3}$$

Next, we relate $\mu = \mathbb{E}_{\pi_{\mathcal{P}_+}}(\text{vdensity}_{\mathcal{P}}(\text{X}))$ to $\text{ovprob}(w_{\mathcal{P}})$.

$$\begin{aligned}\mathbb{E}_{\pi_{\mathcal{P}_+}}(\text{vdensity}_{\mathcal{P}}(\text{X})) &= \sum_{\text{x}\in\mathcal{D}_{\mathcal{P}_+}} \pi_{\mathcal{P}_+}(\text{x}) \cdot \frac{\deg_{\mathcal{P}_+}(\text{x})}{\deg_{\mathcal{P}}(\text{x})} = \sum_{\text{x}\in\mathcal{D}_{\mathcal{P}_+}} \frac{\pi_{\mathcal{P}_+}(\text{x})}{\deg_{\mathcal{P}}(\text{x})} \sum_{\text{q}\in\mathsf{queries}_{\mathcal{P}_+}(\text{x})} 1 \\ &= \sum_{\text{q}\in\mathcal{P}_+} \sum_{\text{x}\in\mathsf{results}(\text{q})} \frac{\pi_{\mathcal{P}_+}(\text{x})}{\deg_{\mathcal{P}}(\text{x})} = \sum_{\text{q}\in\mathcal{P}_+} w_{\mathcal{P}}(\text{q}) = w_{\mathcal{P}}(\mathcal{P}_+)\end{aligned}$$

Since underflowing queries in $\mathcal{P}$ have zero weight, then

$$\begin{aligned}w_{\mathcal{P}}(\mathcal{P}_+) &= 1 - w_{\mathcal{P}}(\mathcal{P}_-) = 1 - \Pr_{w_{\mathcal{P}}}(\text{Q} \in \mathcal{P}_-) \\ &= 1 - \Pr_{w_{\mathcal{P}}}(\text{card}(\text{Q}) > k) = 1 - \text{ovprob}(w_{\mathcal{P}}).\end{aligned}$$

Substituting back in Equation 3, we have:

$$\frac{1}{2} \cdot \text{ndev}_{\pi_{\mathcal{P}_+}}(\text{vdensity}_{\mathcal{P}}(\text{X})) \leq \frac{1 - (1 - \text{ovprob}(w_{\mathcal{P}}))}{1 - \text{ovprob}(w_{\mathcal{P}})} = \frac{\text{ovprob}(w_{\mathcal{P}})}{1 - \text{ovprob}(w_{\mathcal{P}})}.$$

□

B.2 Analysis of query and fetch costs

Theorem 21 (restated) *The query cost of the PB sampler is at most:*

$$\frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot (1 - \text{ovprob}(w_{\mathcal{P}}))} \cdot \left(\frac{|\mathcal{P}|}{|\mathcal{P}_+|} \cdot \frac{k}{\text{avg}_{q\in\mathcal{P}_+} \text{card}(q)} + \text{qcost}(\hat{\pi}) \right).$$

The fetch cost of the PB sampler is at most:

$$\frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot (1 - \text{ovprob}(w_{\mathcal{P}}))} \cdot (1 + \text{fcost}(\hat{\pi})).$$

Proof. Search engine queries are made in two of the subroutines of the PB sampler: (1) In DDSampler, which selects a random document from the results of the query Q that it gets from QCSampler; and (2) In QCSampler, in order to determine the cardinality of the selected random query Q. Queries can also be possibly made in the $\texttt{getWeight}_{\hat{\pi}}(\text{x})$ procedure, when calculating the target weight $\hat{\pi}(\text{x})$ of a document x. Note that DDSampler needs the results of a query Q only after Q has already been processed by QCSampler and thus has already been submitted to the search engine. Therefore, by careful data management, we can keep the results already returned for Q in memory, and hence avoid the additional search engine queries in DDSampler. We assume, then, from now on that DDSampler does not submit queries to the search engine.

In order to analyze the total number of queries made by PBSampler, we define the following random variables: (1) T is the number of iterations made in the loop of the outer procedure. Note that T determines exactly the number of calls to the $\texttt{getWeight}_{\hat{\pi}}(\text{x})$ procedure as well as the number of

calls to the QCSampler procedure. (2) S_i (for $i = 1, 2, \ldots$) is the number of iterations made in the loop of QCSampler during its i-th invocation. The total number of queries submitted to the search engine by the PB sampler is at most $\sum_{i=1}^{T} S_i + T \cdot \text{qcost}(\hat{\pi})$.

The query cost of PBSampler is then $\mathbb{E}(\sum_{i=1}^{T} S_i) + \mathbb{E}(T) \cdot \text{qcost}(\hat{\pi})$. In order to analyze the first term, we resort to Wald's identity. Note that the conditions of Wald's identity are met: $S_1, S_2, \ldots$ are i.i.d. random variables and the event $\{T = i\}$ is independent of $S_{i+1}, S_{i+2}, \ldots$. Hence, the query cost of the PB sampler is $\mathbb{E}(T) \cdot \mathbb{E}(S) + \mathbb{E}(T) \cdot \text{qcost}(\hat{\pi}) = \mathbb{E}(T) \cdot (\mathbb{E}(S) + \text{qcost}(\hat{\pi}))$, where S is the random number of iterations in a single invocation of QCSampler.

In order to bound $\mathbb{E}(T)$, we analyze the parameters of the rejection sampling applied at the outer procedure:

1. The target distribution is $\pi_{\mathcal{P}_+}$. As shown in the proof of Proposition 15, the corresponding normalization constant is: $Z_{\hat{\pi}_{\mathcal{P}_+}} = Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+})$.
2. The trial distribution is $d_{\mathcal{P}_+}$.
3. As shown in the proof of Theorem 18, the trial weight distribution is q. Its normalization constant is: $Z_{\hat{q}} = \deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})$.

As shown in the proof of Theorem 18,

$$\mathbb{E}_{\pi_{\mathcal{P}_+}}\left(\frac{d_{\mathcal{P}_+}(\mathrm{X})}{q(\mathrm{X})}\right) \;=\; \frac{\deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})}{\deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})} \cdot \mathbb{E}_{\pi_{\mathcal{P}_+}}(\text{vdensity}_{\mathcal{P}}(\mathrm{X})).$$

In Theorem 19 we showed:

$$\mathbb{E}_{\pi_{\mathcal{P}_+}}(\text{vdensity}_{\mathcal{P}}(\mathrm{X})) \;=\; 1 - \text{ovprob}(w_{\mathcal{P}}).$$

Therefore, applying Theorem 6, we have:

$$\begin{aligned}
\mathbb{E}(T) &= \frac{C \cdot Z_{\hat{q}}}{Z_{\hat{\pi}_{\mathcal{P}_+}} \cdot \mathbb{E}_{\pi_{\mathcal{P}_+}}\left(\frac{d_{\mathcal{P}_+}(\mathrm{X})}{q(\mathrm{X})}\right)} \\
&= \frac{C \cdot \deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot \frac{\deg_{\mathcal{P}}(\mathcal{D}_{\mathcal{P}_+})}{\deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})} \cdot (1 - \text{ovprob}(w_{\mathcal{P}}))} \\
&= \frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot (1 - \text{ovprob}(w_{\mathcal{P}}))} \qquad (4)
\end{aligned}$$

Next, we bound the expected number of iterations made in the loop of QCSampler. To this end, we analyze the parameters of the rejection sampling applied at QCSampler:

1. The target distribution is $c_{\mathcal{P}_+}$ and its normalization constant is: $Z_{\hat{c}_{\mathcal{P}_+}} = \text{card}(\mathcal{P}_+)$.
2. The trial distribution is the uniform distribution on $\mathcal{P}$ and its normalization constant is: $Z_{\hat{u}_{\mathcal{P}}} = |\mathcal{P}|$. The trial weight distribution is identical to the trial distribution in this case.

3. The envelope constant is $C = k$.

By the analysis of rejection sampling, we have:

$$\mathbb{E}(S) \;=\; \frac{k \cdot |\mathcal{P}|}{\mathrm{card}(\mathcal{P}_+)} \;=\; \frac{|\mathcal{P}|}{|\mathcal{P}_+|} \cdot \frac{k}{\mathrm{avg}_{\mathrm{q} \in \mathcal{P}_+} \mathrm{card}(\mathrm{q})}. \tag{5}$$

Combining the expressions derived at Equations 4 and 5, the total query cost of the PB sampler is at most:

$$\mathbb{E}(T) \cdot (\mathbb{E}(S) + \mathrm{qcost}(\hat{\pi})) \;= \\ \frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot (1 - \mathrm{ovprob}(w_{\mathcal{P}}))} \cdot \left(\frac{|\mathcal{P}|}{|\mathcal{P}_+|} \cdot \frac{k}{\mathrm{avg}_{\mathrm{q} \in \mathcal{P}_+} \mathrm{card}(\mathrm{q})} + \mathrm{qcost}(\hat{\pi}) \right).$$

Next, we analyze the fetch cost of the PB sampler. Pages are fetched in two of the sampler's procedures: (1) In the outer procedure, in order to compute the degrees of documents; and (2) In the $\texttt{getWeight}_{\hat{\pi}}(\mathrm{x})$ procedure, in order to compute the target weight of documents. The number of fetches is determined directly by the number of iterations in the outer procedure, which we denoted by T. Therefore, the fetch cost is at most $\mathbb{E}(T) \cdot (1 + \mathrm{fcost}(\hat{\pi}))$. Using the analysis of $\mathbb{E}(T)$ above, the fetch cost is at most:

$$\frac{C \cdot \deg_{\mathcal{P}_+}(\mathcal{D}_{\mathcal{P}_+})}{Z_{\hat{\pi}} \cdot \pi(\mathcal{D}_{\mathcal{P}_+}) \cdot (1 - \mathrm{ovprob}(w_{\mathcal{P}}))} \cdot (1 + \mathrm{fcost}(\hat{\pi})).$$

□

C Random walk based sampler – Proofs

Theorem 24 (restated) *Let $\varepsilon > 0$. Suppose we run the MH sampler (resp., the MD sampler) with a burn-in period B that guarantees the approximate MH Markov chain (resp., approximate MD Markov chain) reaches a distribution, which is at distance of at most ε from the limit distribution. Then, the sampling bias of the MH sampler (resp., MD sampler) is at most:*

$$\frac{1}{2} \mathrm{ndev}_{\pi_F}(\mathrm{vdensity}_{\mathcal{P}}(X)) + \varepsilon.$$

The sampling recall of the MH sampler (resp., MD sampler) is at least:

$$\pi(F) \cdot (1 - \frac{1}{2} \mathrm{ndev}_{\pi_F}(\mathrm{vdensity}_{\mathcal{P}}(X)) - \varepsilon).$$

Proof. By Theorems 10 and 9, the limit distributions of the MH and the MD samplers are equal to:

$$\eta(\mathrm{x}) = \pi_F(\mathrm{x}) \; \frac{\frac{d_F(\mathrm{x})}{q_F(\mathrm{x})}}{\mathbb{E}_{\pi_F}\left(\frac{d_F(\mathrm{X})}{q_F(\mathrm{X})}\right)}.$$

By Proposition 7, the distance between this limit distribution and the target distribution π_F is bounded as follows:

$$||\eta - \pi_F|| \;\leq\; \frac{1}{2}\,\mathrm{ndev}_{\pi_F}\left(\frac{d_F(\mathrm{X})}{q_F(\mathrm{X})}\right).$$

Now, for every $\mathrm{x} \in F$,

$$\frac{d_F(\mathrm{x})}{q_F(\mathrm{x})} \;=\; \frac{\frac{\deg_{\mathcal{P}_+}(\mathrm{x})}{\deg_{\mathcal{P}_+}(F)}}{\frac{\deg_{\mathcal{P}}(\mathrm{x})}{\deg_{\mathcal{P}}(F)}} \;=\; \mathrm{vdensity}_{\mathcal{P}}(\mathrm{x}) \cdot \frac{\deg_{\mathcal{P}}(F)}{\deg_{\mathcal{P}_+}(F)}.$$

Since normalized mean deviation is invariant under multiplication by scalars, we therefore have:

$$||\eta - \pi_F|| \;\leq\; \frac{1}{2}\,\mathrm{ndev}_{\pi_F}(\mathrm{vdensity}(\mathrm{X})).$$

Consider now the distribution η' obtained after running the approximate MH Markov chain (resp., MD Markov chain) for B steps. η' is the distribution of samples generated by the procedure. Since $B \geq T_\varepsilon(P'_{\mathrm{MH}})$ (resp., $B \geq T_\varepsilon(P'_{\mathrm{MD}})$), then

$$||\eta' - \eta|| \;\leq\; \varepsilon.$$

Therefore, by the triangle inequality,

$$||\eta' - \pi_F|| \;\leq\; \frac{1}{2}\,\mathrm{ndev}_{\pi_F}(\mathrm{vdensity}(\mathrm{X})) + \varepsilon. \tag{6}$$

We next show that this gives also an upper bound on the sampling bias, i.e., on $||\eta'_{\mathrm{supp}(\eta')} - \pi_{\mathrm{supp}(\eta')}||$.

$$\begin{aligned} ||\eta'_{\mathrm{supp}(\eta')} - \pi_{\mathrm{supp}(\eta')}|| \;&=\; \frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\eta')}\left|\eta'(\mathrm{x}) - \frac{\pi(\mathrm{x})}{\pi(\mathrm{supp}(\eta'))}\right| \\ &\leq\; \frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\eta')}\left|\eta'(\mathrm{x}) - \pi_F(\mathrm{x})\right| + \frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\eta')}\left|\pi_F(\mathrm{x}) - \frac{\pi(\mathrm{x})}{\pi(\mathrm{supp}(\eta'))}\right| \end{aligned} \tag{7}$$

Let us bound each of the two above sums separately. To bound the first one, we resort to Equation 6:

$$\begin{aligned} \frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\eta')}\left|\eta'(\mathrm{x}) - \pi_F(\mathrm{x})\right| \;&=\; ||\eta' - \pi_F|| - \frac{1}{2}\sum_{\mathrm{x}\in F\setminus\mathrm{supp}(\eta')}\pi_F(\mathrm{x}) \\ &=\; ||\eta' - \pi_F|| - \frac{1}{2}(1 - \pi_F(\mathrm{supp}(\eta'))). \end{aligned} \tag{8}$$

As for the second sum,

$$\begin{aligned}
&\frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\eta')}\left|\pi_F(\mathrm{x})-\frac{\pi(\mathrm{x})}{\pi(\mathrm{supp}(\eta'))}\right| \\
&= \frac{1}{2}\sum_{\mathrm{x}\in\mathrm{supp}(\eta')}\left|\pi_F(\mathrm{x})-\frac{\pi_F(\mathrm{x})}{\pi_F(\mathrm{supp}(\eta'))}\right| \\
&= \frac{1}{2}\left(\frac{1}{\pi_F(\mathrm{supp}(\eta'))}-1\right)\sum_{\mathrm{x}\in\mathrm{supp}(\eta')}\pi_F(\mathrm{x}) \\
&= \frac{1}{2}\left(\frac{1}{\pi_F(\mathrm{supp}(\eta'))}-1\right)\cdot\pi_F(\mathrm{supp}(\eta')) \\
&= \frac{1}{2}(1-\pi_F(\mathrm{supp}(\eta'))) \qquad (9)
\end{aligned}$$

Combining Equations (6)–(9), we have:

$$\begin{aligned}
||\eta'_{\mathrm{supp}(\eta')}-\pi_{\mathrm{supp}(\eta')}|| &\leq ||\eta'-\pi_F||-\frac{1}{2}(1-\pi_F(\mathrm{supp}(\eta')))+\frac{1}{2}(1-\pi_F(\mathrm{supp}(\eta'))) \\
&= ||\eta'-\pi_F|| \leq \frac{1}{2}\,\mathrm{ndev}_{\pi_F}(\mathrm{vdensity}(\mathrm{X}))+\varepsilon.
\end{aligned}$$

This gives us the desired upper bound on the sampling bias. We now turn to bounding the recall of the MH and MD samplers. Using Equation (6) and the characterization of total variation distance given in Lemma 2, we have:

$$|\eta'(\mathrm{supp}(\eta'))-\pi_F(\mathrm{supp}(\eta'))| \leq \frac{1}{2}\,\mathrm{ndev}_{\pi_F}(\mathrm{vdensity}(\mathrm{X}))+\varepsilon.$$

Since $\eta'(\mathrm{supp}(\eta'))=1$, then this implies:

$$\pi(\mathrm{supp}(\eta')) = \pi(F)\cdot\pi_F(\mathrm{supp}(\eta')) \geq \pi(F)\cdot(1-\frac{1}{2}\,\mathrm{ndev}_{\pi_F}(\mathrm{vdensity}(\mathrm{X}))-\varepsilon).$$

This gives us the lower bound on the recall of the MH and MD samplers. □