Reducing Search Lengths with Locally Precomputed Partial Random Walks

Víctor López Millán
Universidad CEU San Pablo, Spain

Luis López
Universidad Rey Juan Carlos, Spain

Vicent Cholvi
Universitat Jaume I, Spain

Antonio Fernández Anta
Institute IMDEA Networks, Spain

February 17, 2012

Abstract

Random walks can be used to search complex networks for a desired resource. To reduce the number of hops necessary to find resources, we propose a search mechanism based on building random walks connecting together partial walks that have been precomputed at each network node in an initial stage. The resources found in each partial walk are registered in its associated Bloom filter. Searches can then jump over partial nodes in which the resource is not located, significantly reducing search length. However, additional unnecessary hops come from false positives at the Bloom filters. Two variations of the mechanism just described have been considered, differing in the type of partial walks computed in the initial stage: simple random walks or self-avoiding random walks. Analytical models have been developed to predict the expected search length of these mechanisms. When partial walks are random walks, the model also provides expressions for the optimal size of the partial walks and the corresponding optimal (shortest) expected search length. We have found that the optimal search length is proportional to the square root of the expected length of searches based on simple random walks, achieving a significant improvement. Further reductions are obtained when partial walks are self-avoiding random walks. Simulation experiments are used to validate these predictions and to assess the impact of the number of partial walks precomputed in each node. We have found that with just two partial walks per node the results are similar to those obtained for larger values, which is a significant result regarding the practical implementation of the search mechanism.

Keywords: Random walks, self-avoiding random walks, network search, resource location, search length.

1 Introduction

A random walk in a network is a routing mechanism that chooses the next node to visit uniformly at random among the neighbors of the current node. Random walks have been extensively studied in Mathematics, where they have been modeled as finite Markov chains [1 2 3], and have been used in a wide range of applications such as statistic physics, population dynamics, bioinformatics, etc.

When applied to communication networks, it has had a profound impact in algorithms and complexity theory. Some of the advantages of random walks are their simplicity, their small processing power consumption at the nodes, and the fact that they need only local information, avoiding the bandwidth overhead necessary in other routing mechanisms to communicate with other nodes. Random walks are especially useful when there is no knowledge of the structure of the whole network, or when the network structure changes frequently. For these reasons, random walks have been proposed as a base mechanism for multiple network applications, including network sampling, network searching, network construction, and network characterization [1 3 5 6 7 8 9 10 11 12 13 14 15 16 17 18].

In this work, we are concerned with the problem of searching a network for resources held in its nodes, also known as resource location. In particular, we consider a scenario in which all the nodes of a randomly built overlay network may launch independent searches for different resources (e.g., files) at any time, without the help of a centralized server. We consider resources to be randomly placed in the nodes across the network. In this scenario, we are interested in measuring the average performance of searches between any pair of nodes.

If resources are assumed to be unique, the problem is reduced to finding the node that holds the resource, the target node, starting at some source node. Random walks can be used to perform such a search as follows. The source node is checked for the resource. If it is not found locally, the search hops to a random neighbor, checking that node for the resource. The search proceeds through the network in this way until the target node is visited. Due to the random nature of the walk, some nodes may be visited more than once (unnecessarily from the search standpoint), while other nodes may remain unvisited for a long time. The number of hops taken to find the resource is called the search length of that walk. The performance of this direct application of random walks to network search has been studied in [5 19 20 2 18].

Several modifications of the simple random walk behavior described above have been proposed to improve its performance. Da Fontoura Costa and Travieso [21] study the network coverage of three types of random walks: tra-
tional, preferential to unvisited nodes, and preferential to unvisited nodes. Also, Yang [20] studies the search performance of five random walk variations: no-back (NB), no-triangle-loop (NTL), no-quadrangle-loop (NQL), self-avoiding (SA) and high-degree-preferential self-avoiding (PSA). Self-avoiding walks (SAW) are those that try not to visit nodes that have already been visited. Several variations of this idea have been studied, differing in the probability of revisiting a node. Some examples are: strict SAW, true or myopic SAW, and weakly SAW [22][23].

Das Sarma et al. [24] propose a distributed algorithm to obtain a random walk of a specified length \( \ell \) in a number of rounds proportional to \( \sqrt{\ell} \). In the first phase, every node in the network prepares a number of short (random) walks departing from itself. The second phase takes place when a random walk of a given length starting from a given source node is requested. One of the short walks of the source node is randomly chosen to be the first part of the requested random walk. Then, the last node of that short walk is processed. One of its short walks is randomly chosen, and it is connected to the previous short walk. The process continues until the desired length is reached.

Hieungmany and Shiода [25] propose a random-walk-based file search for P2P networks. A search is conducted along the concatenation of hop-limited shortest path trees. To find a file, a node first checks its file list (i.e., an index of files owned by neighbor nodes). If the requested file is found in the list, the node sends the file request message to the file owner. Otherwise, it randomly selects a leaf node of the hop-limited shortest path tree, and the search follows that path, checking the file list of each node in it.

**Contributions** This paper proposes an application to network search of the technique of concatenating partial walks (PW) to build random walks. Two variations of the mechanism are considered, depending on whether the precomputed partial walks are simple random walks (RW) or self-avoiding random walks (SAW). We will refer to the resulting mechanisms as PW-RW and PW-SAW, respectively.

Although the objective in [26] is also resource location, our approach requires nodes only to compute random walks, simpler to compute than their shortest-path-trees. Also, they need more storage space since they keep the pairs resource-owner. Keeping only resources, we are able to use Bloom filters. Another important difference is that our searches jump over partial walks in which the resource is not located, while their searches traverse the selected tree branch, checking the file list of each node in turn.

As mentioned, locally precomputed partial random walks were used in [24] to build a random walk of a given length. On another hand, our objective is to find a resource, so we need searches to proceed until resources are found, resulting in walks of random lengths.

Our mechanisms use Bloom filters [26] to efficiently store the set of resources (not their owners) held by the nodes in each partial walk. The compactness of Bloom filters comes at the price of possible false positives when checking if a given resource is in the partial walk. False positives occur with a probability \( p \), which is taken into account in our analyses.

We provide an analytical model for the PW-RW technique under the assumption that partial walks are always fresh, i.e., not reused in searches. Expressions are given for the expected search length, the optimal size of the partial walks, and for the optimal expected search length. We found that, when the probability of false positives in Bloom filters is small, the optimal expected search length is proportional to the square root of the expected search length achieved by simple random walks searches, in agreement with the results in [24]. Another interesting finding is that the optimal length of the partial walks does not depend on the probability of false positives of the Bloom filters. Our work also includes an analytical model of the PW-SAW mechanism, which predicts the expected search length as a function of the other parameters of the model, including the number of partial walks precomputed by each node.

The predictions of the models are validated by simulation experiments in three types of randomly built networks: regular, Erdős-Rényi, and scale-free. These experiments are also used to compare the performance of the PW-RW and the PW-SAW mechanisms, and to investigate the influence of the number of partial walks per node. For the PW-RW mechanism, we found that the statistical behavior of the search length for as few as two partial walks is very similar to the predictions of the model, which assumes that partial walks are not reused. For the PW-SAW, the analytical model shows that the expected search length does not depend on the number of partial walks per node, in agreement with what was observed in the experiments for PW-RW.

Finally, we have compared the performance of the proposed search mechanisms with respect to random walk searches. For the PW-RW mechanism we have found a reduction in the average search length with respect to simple random walk ranging from around 98% to 88%. For the PW-SAW mechanism such a reduction is even bigger, ranging from 12% to 5% with respect to PW-RW.

### 2 Analytical Model

#### 2.1 Definitions and Assumptions

Let us consider a randomly built network with arbitrary size and topology, whose nodes hold resources randomly placed in them. Resources are unique, i.e., there is a single instance of each resource in the network. The search problem is defined as finding a certain resource, held by one of the nodes (the target node), starting by a certain node (the source node). For each search, the source node and the target node are chosen uniformly at random among all nodes in the network. A search will perform a walk from the source node to the target node according to the mechanism that is defined below. The number of hops to find the resource is the search length. This search length is a random variable that takes different values when independent.

---

1 A round is a unit of discrete time in which every node is allowed to send a message to one of its neighbors. According to this definition, a simple random walk of length \( \ell \) would then take \( \ell \) rounds to be computed.
searches are performed. The **search length distribution** is defined as the probability distribution of the search length random variable. The **expected search length**, derived from the mentioned distribution, is an interesting performance measure of the searching mechanism in a given network.

The search mechanism proposed in this paper, referred to as PW-RW, exploits the idea of efficiently building **total random walks** from **partial random walks** available at each node of the network. It comprises two stages:

1. **Partial random walks construction**: every node \( i \) in the network precomputes a set \( W_i \) of \( w \) random walks in an initial stage before the searches take place. Each of these partial walks has length \( s \), starting at \( i \) and finishing at a node reached after \( s \) hops. Using the PW-RW mechanism, the partial walks computed in this stage are simple random walks (i.e., the next node to be visited is chosen uniformly at random among the neighbors of the current node).

   During the computation of each partial walk in \( W_i \), node \( i \) registers the resources held by the first nodes in the partial walk (from \( i \) to the one before the last node) in a Bloom filter. The last node of the partial walk is excluded from the filter, being included in the filters of the partial walks departing from it. Bloom filters are space-efficient randomized data structures to store sets, supporting membership queries. Thus, the Bloom filter of a partial walk can be queried for a given resource. If the result is negative, the resource is not in any of the nodes of the partial walk. If the result is positive, the resource is in one of the nodes of the partial walk, unless the result was a **false positive**, which occurs with a certain probability \( p \). The size of the Bloom filters can be designed for a target (small) \( p \) considered appropriate.

   A variation of the partial walk construction mechanism consists of using partial walks that are **self-avoiding**. That is, the next node in a walk is chosen uniformly at random among the neighbors that have not been visited so far by that walk (if all neighbors have already been visited, it chooses uniformly at random among all neighbors). The basic idea is to revisit less nodes, thus increasing the chances of locating the desired resource. In Section 4 we will explore such an approach.

2. **The searches**: after the partial walks are constructed, searches are performed in the network in the following fashion. Let a search start at a node \( A \). A partial walk in \( W_A \) is chosen uniformly at random. Its Bloom filter is then queried for the desired resource. If the result is negative, the search **jumps** to node \( B \), the last node of that partial walk. Note that the current node and the node to which the search jumps are not neighbors in the overlay network in general. Jumps therefore make use of the underlying network.

   The process is then repeated at \( B \): a partial walk in \( W_B \) is chosen uniformly at random and its Bloom filter is queried for the resource. The search keeps jumping in this way while the results of the queries are negative. If, when at a node \( C \), the query to the Bloom filter (of a partial walk randomly chosen from \( W_C \)) gives a positive result, the search **traverses** that partial walk looking for the resource. It starts checking if the current node \( C \) has the desired resource. If it does not, the search takes a step to the next node of the partial walk, checking again if it has the resource. The search proceeds through the partial walk in this way until the resource is found or the partial walk is finished. If the resource is found, the search stops. If the search reaches the last node \( D \) of the partial walk without having found the resource in the previous nodes, it means that the result of the Bloom filter query was a false positive. The search then randomly chooses a partial walk in \( W_D \) and decides whether to jump over it or to traverse it depending on the result of the query to its Bloom filter, as described above.

   Therefore, a search can be qualitatively described as a sequence of jumps over partial walks, interleaved with some partial walks traversals due to false positives, and finished by the traversal of a partial walk until the target node is visited. This last partial walk will be incomplete in general, in the sense that its size will be less or equal to \( s \) (see Figure 1).

   In order to increase the success rate of the queries performed at each node, it is possible to check all the partial walks of a node, instead of checking only one. Then, one partial walk can be randomly chosen among those that gave a positive result for the desired resource. However, this approach makes the searching mechanism more prone to choose partial walks with false positives, therefore decreasing its efficiency. As a matter of fact, we have performed some experiments and it has been found that this approach works well only for small values of \( p \). So, in this paper we consider the case where a single partial walk is checked at each node.

We assume that stage 1 is performed initially by all nodes, before the nodes starts launching searches. Afterwards, partial walks can be recomputed to account for changes in the network (nodes added or removed) and in the resources (resources added to, or removed from, nodes). This can be done, for instance, after a predefined number of searches have been performed. Then, the cost of computing the partial walks needs to be added to the cost of performing that number of searches. In what follows we analyze and simulate the searches performed with the initial set of partial walks, without considering the recomputation mechanism. The cost of precomputing the partial walks is analyzed in Section 4.3.

At this point, we emphasize the difference between the search just defined and the total walk that supports it, consisting of the concatenation of partial walks as defined above. Searches are shorter in length than their corresponding total walks because of the number of steps saved in jumps over partial walks in which we know that the

---

2 More concretely, \( p \) is the probability of obtaining a positive result conditioned on the desired resource not being in the filter.

3 In fact, neighbors in the overlay network are not neighbors in general in the underlying network either. Therefore, both jumps and normal steps make use of the underlying network.
resource is not located (although these saving may be reduced by the unnecessary steps due to Bloom filter false positives).

We measure the length of searches in hops, some of which are jumps (over partial walks) and other are steps (traversing partial walks). In turn, we distinguish between trailing steps, if they are the ones taken after a true positive of a Bloom filter (the resource is found), and unnecessary steps, if they are taken after a false positive (the resource is not found). The number of jumps in a search ranges between zero and the number of partial walks in the corresponding total walk, depending on the number of Bloom filters false positives in that search. The definition of the search mechanism and the associated concepts are illustrated by the example in Figure 1 in which partial walks of size $s = 6$ are used.

2.2 The PW-RW Mechanism

We make an additional assumption in order to simplify the analysis of the PW-RW mechanism. Once a partial walk has been used in the total walk of a search, it is never used again in that total walk or in any other searches. In other words, partial walks are always fresh. Thus we guarantee that the total walks are true random walks. This implies that in practice each node needs to have a large number of precomputed partial walks ($w$), assumption that would compromise the benefits of the proposed mechanism in practice. Simulations in Section 3 show that real cases with small $w$ behave very similarly to the base case provided by the model.

Let $L_s$ be the random variable representing the number of hops in the search (i.e., its length). The subscript $s$ is the size (length) of the partial walks used. The expected search length is denoted by $L_s$. Finally, $L$ is defined as the random variable representing the number of hops of the corresponding total walk. Its expected search length is denoted as $L$. Making use of the assumption that partial walks are always fresh (never reused when building a total walk), $L$ can be viewed as the length of a search based on a simple random walk in the considered network, and $L$ as the expected search length of random walks in that network. Then, we can state the following theorem.

**Theorem 1** If the expected number of trailing steps is assumed to be uniformly distributed in $[0, s-1]$, then the expected search length is

$$L_s = \left( \frac{s}{2} + \frac{2L + 1}{2s} - 1 \right) \cdot (1 - p) + L \cdot p.$$  

(1)

Proof. Let $P$, $J$, $U$ and $T$ be random variables representing the number of partial walks, jumps, unnecessary steps and trailing steps in a search, respectively. Their expectations are denoted as $\overline{P}$, $\overline{J}$, $\overline{U}$ and $\overline{T}$. Since hops in a search can be jumps, unnecessary steps or trailing steps, it follows that,

$$L_s = J + U + T.$$  

(2)

Then, the expected search length for partial walks of size $s$ is

$$\overline{L}_s = \overline{J} + \overline{U} + \overline{T}.$$  

(3)

The expected number of jumps can be obtained from the expected number of partial walks in the search ($\overline{P}$) and from the probability of false positive ($p$);

$$\overline{J} = \overline{P} \cdot (1 - p),$$  

(4)

since $J$ follows a binomial distribution $B(P, 1 - p)$, where the number of experiments is the random variable representing the number of partial walks in a search ($P$) and the success probability is the probability of obtaining a negative result in a Bloom filter query ($1 - p$). Also, for the expected number of unnecessary steps:

$$\overline{U} = \overline{P} \cdot p \cdot s,$$  

(5)

since $\overline{P} \cdot p$ is the expected number of false positives in the search and each of them contributes with $s$ unnecessary steps.

The number of partial walks in a search can be obtained dividing the length of the total walk by the size of a partial walk: $P = \left\lfloor \frac{L}{s} \right\rfloor = \frac{L - T}{s}$. Then, the expected number of partial walks in a search is:

$$\overline{P} = \frac{L - T}{s}.$$  

(6)

Since we assume that the expected number of trailing steps is uniformly distributed between 0 and ($s - 1$), its expectation is:

$$\overline{T} = \frac{s - 1}{2}.$$  

(7)

Using Equations 4 to 7 in Equation 3 we have:

$$L_s = \left( \frac{s}{2} + \frac{2L + 1}{2s} - 1 \right) \cdot (1 - p) + \left( \frac{L - \left( \frac{s}{2} + \frac{2L + 1}{2s} - 1 \right)}{2} \cdot p \right) \cdot s.$$  

(8)

where the first term is the expected length of the search length for a “perfect” Bloom filter (one that never returns a false positive when the resource is not in the filter, i.e., $p = 0$), slightly higher probability than longer ones. This can be shown analytically and has been confirmed in our experiments (see Appendix A). Therefore, the expected value in our analysis, derived analytically and has been confirmed in our experiments (see Appendix A). Therefore, the expected value in our analysis, derived from a perfectly uniform distribution, is slightly higher than the real average value.

In the following, we make implicit use of the linearity properties of expectations of random variables.

6If $Y$ is a random variable with a binomial distribution with success probability $p$, in which the number of experiments is in turn the random variable $X$, it can be easily shown that $\overline{Y} = \overline{X} \cdot p$ (see Appendix A).
and the second term is the expectation of the additional search length due to false positives \((p \neq 0)\). Another interpretation of this expression is obtained if we reorganize it to make explicit the contributions of a perfect filter and of a “broken” filter (one that always returns a false positive result when the resource is not in the filter, i.e., \(p = 1\)) as follows.

\[
T_s = \left(\frac{s}{2} + \frac{2L + 1}{2s} - 1\right) \cdot (1 - p) + L \cdot p.
\]

From the above Theorem and using Calculus on the coefficient of \((1 - p)\) in Equation\(^\text{[1]}\) (taking into account that all dependencies on \(s\) are concentrated in it), we have:

**Corollary 2** The optimal size of the partial walks, i.e., the size of the partial walks that minimizes the expected search length, is:

\[
s_{\text{opt}} = \sqrt{2L} + 1.
\]

The obtained value needs to be rounded to the an integer, which is omitted in the notation. Observe that the optimal size of the partial walks is independent from the probability of false positives in the Bloom filters, while the expected search length \((T_s)\) does of course depend on it.

**Corollary 3** The optimal expected search length, i.e., the expected search length when partial walks of optimal size are used, is:

\[
T_{\text{opt}} = \left(\sqrt{2L} + 1 - 1\right) \cdot (1 - p) + L \cdot p = (s_{\text{opt}} - 1) \cdot (1 - p) + L \cdot p.
\]

This result is an interesting relation between the optimal length of the search and the optimal length of the partial walks. If we consider perfect Bloom filters \((p = 0)\), we have \(T_{\text{opt}} = s_{\text{opt}} - 1\), which for large \(L\) (e.g., for large networks) becomes \(T_{\text{opt}} \approx s_{\text{opt}}\). Therefore, we have found that, for large \(N\) and \(p = 0\), the optimal expected search length approximately equals the optimal length of the partial walks. For arbitrary values of \(p\), Equation\(^\text{[11]}\) shows that \(T_{\text{opt}}\) is linear in \(p\).

### 2.3 Cost of Precomputing Partial Walks

Since searches use the partial walks precomputed by each of the nodes of the network, the cost of this computation must be taken into account. We measure this cost as the number of messages \(C_p\) that need to be sent to compute all the partial walks in the network. Observe that \(C_p\) is independent from other factors like the processing power of nodes, the bandwidth of links and the load of the network. This number can be simply obtained as \(C_p = N \cdot w \cdot (s_{\text{opt}} + 1)\), since each of the \(N\) nodes in the network computes \(w:\) partial walks, sending \(s_{\text{opt}}\) messages to build each of them plus one extra message to get back to its source node. Note that we are assuming here that partial walks of optimal size, as defined in Equation\(^\text{[11]}\), are used.

We can compare the cost of precomputing random walks (i.e., \(C_p\)) with the expected cost of searches themselves \(C_s\), which is defined as the number of messages needed to perform them.

Let us suppose that each node starts \(b\) searches that are processed by the network with the set of partial walks precomputed initially. When using optimal length partial walks, searches have an expected length \(T_{\text{opt}}\). Since an extra message is needed to report the search success to the starting node, the total number of messages can be written as \(C_s = N \cdot b \cdot (T_{\text{opt}} + 1)\). For large networks and low values of \(p\), we have that \(T_{\text{opt}} \approx s_{\text{opt}}\) (see Corollary\(^\text{3}\)). Therefore, \(C_s \approx N \cdot b \cdot (s_{\text{opt}} + 1)\).

Now, we compare the cost of precomputing random walks with the cost of searches themselves simply by obtaining \(C_p/C_s \approx w/b\). This relative cost can be made as low as desired by setting the number of searches \(b\) to a value large enough with respect to the number of partial walks per node, \(w\).

Finally, we do note that the repetition of the partial random walk construction (stage 1) could overlap in time with the searches (stage 2), and that the partial walks of a node (and of all nodes) could also be precomputed in parallel.

### 3 Performance Evaluation

The goal of this section is to apply the model presented in the previous section to real networks, and to validate its predictions with data obtained from simulations. Three types of networks have been chosen for the experiments: regular networks (constant node degree), Erdős-Rényi (ER) networks and scale-free networks (with power law on the node degree). A network of each type and size \(N = 10^4\) has been randomly built with the method proposed by Newman et al.\(^\text{[27]}\) for networks with arbitrary degree distribution, setting their average node degree to \(k = 10\). Each network is constructed in three steps: (1) a preliminary network is constructed according to its type; (2) its degree distribution is extracted, and (3) the final network is obtained feeding the referenced method with that degree distribution. For each experiment, \(10^6\) searches have been performed, with the source and target nodes chosen uniformly at random among the \(N\) nodes.

#### 3.1 Optimal Partial Walk Size and Expected Search Length in PW-RW

We start by applying the result in Theorem\(^\text{[1]}\) to the networks described above to obtain the expected search length as a function of the size of the partial walks.\(^\text{5}\)

Figure\(^\text{2}\) provides a plot of the expected search length \((T_s)\) given by Equation\(^\text{[1]}\) as a function of the size of the partial walks \((s)\), when the probability of a false positive

\(^5\)For each network, the expected length of a random walk search \((T)\) is needed. We estimate these expected values by simulating \(10^6\) simple random walk searches and averaging their lengths in each of the networks (these average search lengths are denoted using lower-case \((\bar{T})\) to distinguish them from the actual expected value \((T)\) in the model. The values obtained from the experiments are: \(T_{\text{avg}} = 11246, T_{\text{ER}} = 12338,\) and \(T_{\text{sf}} = 15166\).
in the Bloom filter is set to $p = 0$. The curves for the three networks show a minimum point $(s_{\text{opt}}, \mathcal{L}_{\text{opt}})$. This behavior is due to the fact that, when $s$ is small, the number of jumps needed to reach a partial walk containing the chosen resource grows, therefore increasing the value of $\mathcal{L}_{\text{opt}}$. In turn, for larger values of $s$, the number of trailing steps within the last partial walk grows, also increasing the value of $\mathcal{L}_{\text{opt}}$ (see Equation 5).

Figure 3 illustrates (using the result in Corollary 3 and taking into account the fact that $s_{\text{opt}}$ is independent from the value of $p$) the optimal expected search length ($\mathcal{L}_{\text{opt}}$) as a function of the probability of false positives ($p$). It can be seen that it grows linearly: the regular network exhibits the smallest slope, followed by the ER network and then by the scale-free network. For $p = 1$, Equation 11 degenerates to $\mathcal{L}_{\text{opt}} = \mathcal{L}$, since the search performs all the hops of the total walk (i.e., it is a random walk). In fact, Equation 11 also degenerates to $\mathcal{L}_p = \mathcal{L}$ in this case, meaning that the expected search length is that of random walk searches regardless the size of the partial walks ($s$).

### 3.2 Distributions of Search Lengths in PW-RW

The aim of this section is to experimentally explore how the use of partial walks affects the statistical distribution of search lengths.

**Length distributions** We first obtain the lengths distributions of searches using partial walks that are always fresh (i.e., never reused). Later in this section we will discuss the effect of having a limited number of partial random walks that are reused. We consider each random walk to be the total walk of a search based on partial walks. For each original random walk, we break it in pieces of size $s$, which are taken as the partial walks that make up the total walk. Then we consider a search that uses those partial walks and count the number of hops (jumps plus trailing steps plus unnecessary steps). This gives the length of the search if it had been constructed using those (precomputed) partial walks. Note that the partial walks are not reused because they are obtained from independent (real) random walks.

The search length distributions in the regular network for $p = 0$ and for several values of $s$ are shown in Figure 4(a). The plots also show, as vertical bars, the average lengths computed from each distribution. These average values are very close to the expected values calculated with Equation 1 ($\mathcal{L}_{50} = 248.9, \mathcal{L}_{150} = 149.0$ and $\mathcal{L}_{1000} = 510.2$). Therefore, our model accurately predicts average lengths of searches based on partial walks of size $s$ in the three types of networks considered in our experiments.

As for the shape of the distributions, we observe that for low $s$ ($s = 50$ in Figure 4(a)) the search lengths are dominated by the number of jumps, which is proportional to the length of the total walk. On the other hand, for high $s$ ($s = 1000$ in Figure 4(a)) the distribution adopts a rather uniform shape. Search lengths are dominated here by the number of trailing steps in the last partial walk, and this has approximately an uniform distribution between 0 and $s - 1$, as mentioned earlier. The optimal length for the partial walks, $s_{\text{opt}}$ ($s = 150$ in Figure 4(a)), represents a transition point between these two effects. The shape is such that the values around the average search length (which approximately equals $s_{\text{opt}}$, according to Equation 11) are also the most frequent.

Once it has been found the optimal length for the partial walks $s_{\text{opt}}$ (which is known to be independent of the value of $p$), we investigate the effect of the probability of false positive of Bloom filters in these distributions. Figure 4(b) shows the distributions of search lengths (histograms) for the regular network when $s = s_{\text{opt}}$ and for several values of $p$. It can be seen that the distributions get wider and lower as $p$ grows, pushing average search lengths to higher values, in accordance with Figure 3. However, we observe that the most frequent lengths remain the same regardless of the value of $p$. For $p = 0$, the most frequent value for each network approximately equals the average search length which, in turn, approximately equals the optimal length of the partial walks ($s_{\text{opt}} = 150$ for the regular network). For greater values of $p$, the average search length...
(a) Search lengths for $p = 0$ and for $s = s_{opt} = 150$, $s = 50$ and $s = 1000$.

(b) Search lengths for $s_{opt}$ and for $p = 0, 0.01, 0.1$.

Figure 4: Distributions of search lengths (histograms) using always fresh partial walks in the regular network.
grows while the most frequent value stays the same.

Regarding the distributions for the ER and the scale-free networks, they have similar shapes and are not shown here. However, we have used these distributions to obtain Table 1 (explained below).

**Effect of using partial walks** At this point, we note that we have been assuming that partial walks are always fresh. However, in practical scenarios it seems quite reasonable to consider a limited number of partial random walks that are reused. In Appendix C we have explored the distributions of search lengths when the total walks are built reusing a limited number \( w \) of partial walks precomputed in each node. As it can be readily seen there, we conclude that, for the types of networks in our experiment, just two precomputed partial walks per node are enough to obtain searches whose lengths are statistically similar to those that would be obtained with always fresh partial walks. So, we can say that our results using fresh partial walks are also valid when using a limited number of partial random walks that are reused.

**Comparison of performance with respect to random walks** Finally, in Table 1 we compare the performance of the proposed search mechanism with respect to random walk searches. We can see that the reduction in the average search length that PW-RW achieves with respect to simple random walk is lower for higher \( p \), ranging from around 98% in the case when \( p = 0 \) to 88% when \( p = 0.1 \). Furthermore, we also see that the achieved reductions are independent of the network type.

### Table 1: Reduction of the average search length achieved by PW-RW with respect to random walk searches.

| Network type | \( p = 0 \) | \( p = 0.01 \) | \( p = 0.1 \) |
|--------------|-------------|-------------|-------------|
| Regular      | 98.67       | 97.68       | 88.73       |
| ER           | 98.71       | 97.68       | 88.42       |
| Scale-free   | 98.83       | 97.79       | 88.43       |

### 4 Self-Avoiding Random Walks (PW-SAW)

As it was pointed when we introduced the "standard" partial random walk construction (Section 2.1), a possible variation of the PW-RW searching mechanism could consist of using partial walks that are self-avoiding. The basic idea is to revisit less nodes, thus increasing the chances of locating the desired resource. The rest of the operation is common for the two proposed mechanisms.

In short, a RW chooses the next node to be visited uniformly at random among the neighbors of the current node, while a SAW chooses the next node uniformly at random among the neighbors that have not been visited so far by that walk. If all neighbors have already been visited, it chooses uniformly at random among all neighbors.

#### 4.1 The PW-SAW Mechanism

When partial walks are self-avoiding walks, their concatenation is not a random walk, and hence the analysis in the previous section is no longer valid. Here we use a different approach, writing a recurrence equation for the expected length, given that the search is currently in any of the nodes it visits. Since we have defined the expected search length for any pair of source and target nodes, the expected length of the search from the current node and the expected length of the search from the source node are the same. Denoting it by \( \mathcal{T}_s \), as in the previous section, we can write:

\[
\mathcal{T}_s = (\mathcal{T}_s + 1) \cdot p_n + (\mathcal{T}_s + s) \cdot p_{fp} + \frac{s-1}{2} \cdot p_{tp},
\]

where \( p_n, p_{tp}, \) and \( p_{fp} \) are the probabilities that the query of the Bloom filter of the chosen partial walk in the current node returns a (true) negative, a true positive, and a false positive result, respectively, with \( p_n + p_{tp} + p_{fp} = 1 \). Solving for \( \mathcal{T}_s \), we obtain:

\[
\mathcal{T}_s = \frac{1}{p_{tp}} \cdot (p_n + s \cdot p_{fp}) + \frac{s-1}{2}.
\]

This equation can be rewritten as:

\[
\mathcal{T}_s = \frac{1 - p_{tp}}{p_{tp}} \cdot \left( \frac{p_n}{1 - p_{tp}} + s \cdot \frac{p_{fp}}{1 - p_{tp}} \right) + \frac{s-1}{2},
\]

which is an alternative formulation of the expected search length, in terms of the expected number of partial walks of the search (\( \mathcal{T} \), as defined in Section 2.2). Note that \((1-p_{tp})/p_{tp}\) is the expectation of \( \mathcal{T} \), a geometric random variable representing the number of failures before a Bloom filter returns a true positive (with probability \( p_{tp} \)). The fractions within the parenthesis are, respectively, the probabilities of jumping a partial walk or traversing it, conditional on the fact that the Bloom filter does not return a true positive. Therefore, the terms in the parenthesis are the expectations of \( \mathcal{T} \) and \( \mathcal{T}_s \), binomial random variables representing the number of jumps and the number of partial walks that are unnecessarily traversed, respectively, as defined in Section 2.2.

We now calculate the probabilities in the equations above using \( P(i,j) \), the probability that, in the \( w \) partial walks of a node, there are \( i \) partial walks that contain the node that holds the resource (i.e., their Bloom filters return a true positive), and \( j \) partial walks that do not contain the resource, but whose filters return false positives:

\[
P(i,j) = B(w,p_r,i) \cdot B(w-i,p,j),
\]

where \( B(m,q,n) \) is the coefficient of the binomial distribution: \( B(m,q,n) = \binom{m}{n} \cdot q^n \cdot (1-q)^{(m-n)} \).

In Equation 15 we are using \( p_r \), defined as the probability that a partial walk includes the node that holds the desired resource. This probability is proportional to the degree of the node that holds the resource, since the probability that a random walk visits a node depends on its degree (see [4], for example). We assume known the number of nodes of each degree \( k \) in the network, i.e., its degree distribution, which we denote by \( n_k \).
Denoting by $k$ the degree of the node that holds the resource, the probability that a partial walk of size $s$ contains the resource is then $p_r(k)$, and it can be estimated as:

$$p_r(k) = 1 - \prod_{i=0}^{s-1} \left( 1 - \frac{k}{S - i k} \right), \quad (16)$$

where $S$ denotes the number of endpoints in the network $(S = \sum_k k n_k)$ and $\bar{k}$ denotes the average degree of the network $(\bar{k} = \sum_k k n_k/N)$. Each factor in the product in Equation (16) represents the probability that the resource is not found in the $i$th hop of a partial walk, conditional on the fact that it was not found in the previous hops of that partial walk. Note that the fraction $k/(S - i k)$ is the probability of the $i$th hop finding the resource, expressed as the number of endpoints that belong to the node that holds the resource divided by the total number of endpoints in the network, except those belonging to nodes already visited by the partial walk, which are $\bar{k}$ per hop, on the average.

Now we rewrite Equation (15) making its dependence on $k$ explicit:

$$P(i, j|k) = B(w, p_r(k), i) \cdot B(w - i, p, j), \quad (17)$$

Then, the probabilities in Equations (12) and (13) are:

$$p_{tp}(k) = \sum_{i=1}^{w} \sum_{j=0}^{w-i} P(i, j|k) \cdot \frac{i}{w}$$

$$p_{fp}(k) = \sum_{i=0}^{w} \sum_{j=1}^{w-i} P(i, j|k) \cdot \frac{j}{w}$$

$$p_n(k) = 1 - p_{tp}(k) - p_n(k). \quad (18)$$

The expected search length can be finally obtained weighing Equation (13) with the probability that the resource is in a node with degree $k$, which is $n_k/N$, for all values of $k$:

$$T_s = \frac{1}{N} \sum_k n_k \left( \frac{1}{p_{tp}(k)} \cdot (p_n(k) + s \cdot p_{fp}(k)) + \frac{s - 1}{2} \right). \quad (19)$$

4.2 Expected Search Length in PW-SAW

In this section, we compare the analytic results from the model with experimental data from simulations. Figure 5 shows the expected search length ($T_s$) as a function of the size of partial walks ($s$) in a regular network, an ER network and a scale-free network, for $p = 0$. The curves in this graph are plotted using Equation (19) and previous equations.

According to the results computed using the PW-SAW model, the minimum search lengths occur for values around $s = 141$, $s = 149$ and $s = 167$ for the regular, ER and scale-free networks, respectively. These values are slightly lower than the ones predicted by the PW-RW model (Figure 2), which were $s_{opt} = 150, 157$ and 174, respectively.

Both the model curves and the simulation experiments have been computed for $w = 5$, chosen as a reference value.

However, it has been observed that very similar results are obtained if we change the value of $w$. Furthermore, plots of the model equations for different values of $w$ are coincident. This behavior was also observed for PW-RW (Section 3.2), where we found that the average search length remained almost constant as we increased $w$. The reason for this is that the probability of the resource being in the chosen partial walk ($p_r$ in Equation (13)) does not depend on the number of partial walks in the node.

We now compare the results of the PW-RW and PW-SAW mechanisms. Figure 6 shows results for PW-RW (left part) and for PW-SAW (right part), in the three networks considered in our study, and for values of $p = 0, 0.01$ and 0.1. Expected search lengths from the analytical models are shown as vertical bars, while average search lengths from the simulations experiments are shown as points. The partial walks size has been set to $s = 150, 157$ and 174 for the regular, ER and scale-free networks, respectively, which are the optimal values predicted by the PW-RW model. For all the networks, we have found a very good correspondence between model predictions and simulation results.

Comparison of performance with respect to PW-RW If we compare the performance of the proposed search mechanisms, we observe that the reduction in the average search length that PW-SAW achieves with respect to PW-RW for a given $p$ is largest for the scale-free network, followed by the ER network and then by the regular network. For each network type, the reduction is larger for higher $p$. Actual values can be found in Table 2.

5 Conclusions

We have proposed two mechanisms to search a network for a desired resource. Both mechanisms are based on building a total walk with partial walks that are precomputed and available at each network node. A Bloom filter for each partial walk stores the resources held by the
nodes in the partial walk, so that the search can jump over partial walks in which the desired resource is not located. The mechanism PW-RW uses simple random walks as partial walks, whereas the mechanism PW-SAW uses self-avoiding walks as partial walks. We have presented analytical models for both mechanisms, and performed simulation experiments to validate their predictions. The mechanism PW-RW achieves a search length proportional to the square root of that obtained by simple random walk searches, when the probability of a Bloom filter returning a false positive is small. We have found that just two partial walks per node are enough to obtain a statistical behavior similar to that of a true random walk built with always fresh partial walks. The mechanism PW-SAW achieves further reductions of the expected search length, which depend on the type of network and the probability of false positives in Bloom filters.

An interesting future work line for this study is to measure the improvement in the search length that can be obtained by using different strategies to choose one of the partial walks available in a node. Another possibility to shorten search lengths is to use more intelligent (and more costly) variants of random walks instead of simple random walks.

Table 2: Reduction of the average search length achieved by PW-SAW with respect to PW-RW.

| Network type | $p = 0$ | $p = 0.01$ | $p = 0.1$ |
|--------------|---------|------------|-----------|
| Regular      | 5.67    | 8.22       | 11.24     |
| ER           | 6.25    | 9.10       | 11.88     |
| Scale-free   | 6.53    | 9.75       | 12.65     |

Table 2: Reduction of the average search length achieved by PW-SAW with respect to PW-RW.

References

[1] Barry D. Hughes. *Random walks and random environments*, volume 1. Clarendon Press, Oxford, 1995.

[2] L. Lovász. Random walks on graphs: a survey. In *Combinatorics, Paul Erdős is eighty*, volume 2, pages 1–46. Keszthely, Hungary, 1993.

[3] Mark E. J. Newman, Albert-László Barabási, and Duncan J. Watts. *The structure and dynamics of networks*. Princeton University Press, 2006.

[4] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random-walks in peer-to-peer networks: algorithms and evaluation. *Performance Evaluation*, 63:241–263, 2006.

[5] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physical Review E*, 64(046135), 2001.

[6] Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. Statistical properties of sampled networks. *Physical Review E*, 73(016102), 2006.

[7] Bosiljka Tadić. Adaptive random walks on the class of web graphs. *European Physical Journal B*, 23(221), 2001.

[8] Santo Fortunato and A Flammini. Random walks on directed networks: the case of pagerank. *e-print physics*, (0604203), 2006.

[9] Y. Chawathe, S. Ratnasamy, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 407–418, Karlsruhe, Germany, 2003.

[10] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the Sixteenth International Conference on Supercomputing*, pages 84–95, New York, New York, United States, 2005.

[11] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Revised papers from the First International Workshop on Peer-to-Peer Systems*, pages 94–103, Cambridge, United States, 2002.

[12] N Bisnik and A. Abouzeid. Modeling and analysis of random walk search algorithms. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P 2005)*, pages 95–103, New York, New York, United States, 2005. IEEE Computer Society.

[13] I. Mabrouki, X. Lagrange, and G. Froc. Random walk based routing protocol for wireless sensor networks. In *Proceedings of the Second International Conference on Performance Evaluation Methodologies and Tools (ValueTools ’07)*, pages 1–10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[14] M. Alanyali, V. Saligrama, and O. Savas. A random-walk model for distributed computation in energy-limited networks. In *Proceedings of the First Workshop on Information Theory and its Application*, 2006.
[15] S. Dolev, E. Schiller, and J. L. Welch. Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(7):893–905, 2006.

[16] C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *Proceedings of the Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, volume 3, pages 2133–2143, 2003.

[17] N. Sadagopan, B. Krishnamachari, and A. Helmy. Active query forwarding in sensor networks. *Ad hoc Networks*, 3:91–113, 2005.

[18] Luis Rodero-Merino, Antonio Fernández Anta, Luis López, and Vicent Cholvi. Performance of random walks in one-hop replication networks. *Computer Networks*, 54(5):781–796, 2010.

[19] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS ’02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[20] Shi-Jie Yang. Exploring complex networks by walking on them. *Physical Review E*, 71(016107), 2005.

[21] Luciano da Fontoura Costa and Gonzalo Travieso. Exploring complex networks through random walks. *Physical Review E*, 75(016102), 2007.

[22] D. Amit, G. Parisi, and L. Peliti. Asymptotic behavior of the ‘true’ self-avoiding walk. *Physical Review B*, 27:1635–1645, 1983.

[23] Gordon Slade. The self-avoiding walk. In *Surveys in Stochastic Processes, Proceedings of the 33rd SPA Conference*. EMS Series of Congress Reports, 2009. in preparation.

[24] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Efficient distributed random walks with applications. In *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, PODC ’10, pages 201–210, New York, NY, USA, 2010. ACM.

[25] Phouvieng Hieunmany and Shigeo Shioda. Characteristics of random walk search on embedded tree structure for unstructured p2ps. *Parallel and Distributed Systems, International Conference on*, 0:782–787, 2010.

[26] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[27] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64(026118), 2001.

---

**Figure 7:** Distributions of the number of trailing steps in the regular network.

### A Distributions of the Number of Trailing Steps

The analysis in Section 2.2 assumes that the distribution of the number of trailing steps in the last partial walk until the search finds the resource is uniform between 0 and \( s - 1 \), corresponding to the cases where the first node/last node in the partial walk holds the desired resource. Recall that the Bloom filter stores the resources held by the \( s \) first nodes in the partial walk, from the node that precomputed the partial walk to the one before its last node (which is included in the partial walks departing from it). We have obtained that distribution from the \( 10^6 \) searches in our experiment for each of the three networks. Figure 7 shows the distributions for the regular network when \( s = 10 \), \( s = s_{opt} = 150 \) and \( s = 1000 \). Distributions for the ER and scale-free networks are similar in shape.

It is observed that there is a slight decrease on the frequency as the number of steps grows. This is due to the fact that the number of trailing steps is essentially the length of the total walk modulus the length of partial walks (\( s \)). The total walk is a random walk, and its distribution can be obtained approximately by Equation 20. Since it is a decreasing function, as it is shown below, the frequency on the left end of an interval of width \( s \) is always higher than the frequency on the right end, thus accounting for the observed decrease.

This means that the analysis in Section 2.2 is pessimistic, since the estimated average number of trailing steps is slightly higher than the real one. Results in Section 3 have shown that values of average search lengths predicted by Equation 1 are very similar to values computed from simulations, with larger error for higher values of \( s \).

The probability distribution of simple random walk searches can be estimated using Equation 20. It can be demonstrated that it is strictly decreasing, that is:

---

*The distribution of simple random walk searches has also been obtained experimentally, showing that Equation 20 is a good approximation.*
Then we have that:

\[ P_i = \left(1 - \sum_{j=0}^{i-1} P_j\right) \cdot \frac{1}{N-1}, \quad \text{for } i > 0. \]

First, it is shown by induction that \( 0 < \sum_{i=0}^{k} P_i < 1 \) for \( k \geq 0 \) and \( N > 0 \). It holds trivially for \( k = 0 \). Then, it is also true for \( k > 0 \) if it holds for \( k - 1 \):

\[
\sum_{i=0}^{k} P_i = \sum_{i=0}^{k-1} P_i + \left(1 - \sum_{i=0}^{k-1} P_i\right) \cdot \frac{1}{N-1}
\]

\[
= \frac{N - 2}{N - 1} \sum_{i=0}^{k-1} P_i + \frac{1}{N - 1}
\]

\[
< \frac{N - 2}{N - 1} + \frac{1}{N - 1} = 1.
\]

Next, it is shown that \( 0 < P_i < 1 \) for \( i \geq 0 \) as a corollary of the previous result. It is checked for \( i = 0 \) by inspection. For \( i > 0 \), we have that:

\[ P_i = \left(1 - \sum_{j=0}^{i-1} P_j\right) \cdot \frac{1}{N-1}. \]

Then we have that:

\[ 0 < 1 - \sum_{j=0}^{i-1} P_j < 1, \]

and:

\[ 0 < P_i = \left(1 - \sum_{j=0}^{i-1} P_j\right) \cdot \frac{1}{N-1} < 1. \]

Finally, it is shown that \( P_i - P_{i-1} < 0 \) for \( i > 0 \). For \( i = 1 \), it is checked by inspection. For \( i > 1 \):

\[
P_i - P_{i-1} = \left(1 - \sum_{j=0}^{i-1} P_j\right) \cdot \frac{1}{N-1} - \left(1 - \sum_{j=0}^{i-2} P_j\right) \cdot \frac{1}{N-1}
\]

\[
= \frac{P_{i-1}}{N - 1}.
\]

Since we have shown that \( 0 < P_{i-1} < 1 \), it follows that \( P_i - P_{i-1} < 0 \).

### B Expectation of a Random Variable with a Binomial Distribution in Which the Number of Experiments is Another Random Variable

Let \( X \) be a random variable with sample space \( S = \mathbb{N}_0 = \{0, 1, 2, \ldots\} \). Let \( Y \) be a random variable representing the number of successes when \( X \) experiments are performed with a success probability \( p \). \( Y \) has a binomial probability distribution \( Y \sim B(X, p) \), where the number of experiments is, in turn, a random variable. Then, from the definition of expectation and applying the Total Probability Theorem, the expectation of \( Y \) is \( E[Y] = E[X] \cdot p \).

\[
E[Y] = \sum_{y=0}^{\infty} y \cdot P_r[Y = y]
\]

\[
= \sum_{y=0}^{\infty} y \cdot \left( \sum_{x=0}^{\infty} P_r[Y = y|X = x] \cdot P_r[X = x] \right)
\]

\[
= \sum_{x=0}^{\infty} E[Y|X = x] \cdot P_r[X = x]
\]

\[
= \sum_{x=0}^{\infty} x \cdot p \cdot P_r[X = x] = E[X] \cdot p.
\]

### C Searches based on reused partial walks

In this section, we explore the distributions when the total walks are built reusing a limited number \( w \) of partial walks precomputed in each node. This is in contrast with our initial assumption that precomputed partial walks are not reused in searches. Here, we attempt to answer the question “How many partial walks does a node need to precompute, for the search lengths distribution to be similar to that corresponding to never reusing partial walks?”.

Our results show that, for the networks considered in our experiment, and for the optimal partial walk size \( s_{opt} \), it is enough to have as few as two precomputed partial walks in every node. The extreme case of having just one precomputed partial walk yields a significant fraction of unfinished searches, since it is relatively easy to build walks that are loops that do not visit all the nodes. Indeed, if the last node of a partial walk is a node whose (only) partial walk has been previously used in that total walk, it will take the search to the same place again, resulting in a never-ending loop. However, if a node has several partial walks, and the search chooses one randomly among them (for the next jump or partial walk traversal), the chances of entering a loop are very small.

Figures [a] to [c] show the search lengths distributions in the regular network. The top plots of these figures show the length distributions of searches based on always fresh partial walks. The middle and bottom plots show the length distributions of searches based on reusing a single partial walk or two partial walks per node, respectively.

We note that the shape of the distributions is the same for all values of \( w \). However, distributions for \( w = 1 \) are lower, and the average search length (marked as a vertical bar) is also smaller. This is due to a significant percentage of unfinished searches (about 26.3%), left out of the histograms, due to loops as explained above. If we focus now on the distributions for \( w = 2 \), we observe that both the distribution and the average search length are very similar to those for always fresh partial walks. We have performed additional experiments with higher values of \( w \), confirming this observation. This suggests that just two precomputed partial walks per node are enough to obtain a behavior close to the theoretical case of using always fresh precomputed partial walks. The distributions of searches in the ER network and the scale-free network are omitted.
here, since their shape and the conclusions drawn are the same as for the regular network.

We now measure the difference between the search length distributions for several values of \( w \) and the base case of always fresh partial walks. In Figure 9 we plot these (signed) differences for \( w = 2 \) and several values of \( p \) in the regular network. It is observed that differences are small for low values of \( p \), growing as \( p \) gets bigger. But the magnitude of the differences seem to be within the order of variation of the values of the histograms for all values of \( p \). As a global measure of the difference between the distributions for \( w = 2 \) and for always fresh partial walks we compute the mean relative difference as

\[
\frac{1}{L_{90\%} + 1} \sum_{\ell=0}^{L_{90\%}} \frac{|h_w(\ell) - h_{af}(\ell)|}{h_{af}(\ell)},
\]

where \( h_w(\ell) \) is the number of searches with length \( \ell \) when using \( w \) partial walks per node, and \( h_{af}(\ell) \) corresponds to the case of always fresh partial walks. The tail of long searches with low frequency is removed from the calculation, since those values yield high relative differences that distort the measurement. For this, the summation includes 90\% of the searches, from length zero up to \( L_{90\%} \), where \( L_{90\%} \) is the 90\% percentile of search lengths. The mean relative differences for \( p = 0 \), \( p = 0.01 \) and \( p = 0.1 \) are, respectively, 0.023, 0.035 and 0.076.

Therefore we conclude that, for the types of networks in our experiment, just two precomputed partial walks per node are enough to obtain searches whose lengths are statistically similar to those that would be obtained with always fresh partial walks.
Figure 8: Search length distributions for always fresh partial walks, $w = 1, 2$ and $p = 0, 0.01, 0.1$ in the regular network.

Figure 9: Difference between search length distributions for $w = 2$ and for always fresh partial walks in the regular network.