# On Efficiency in Searching Networks

Hsinping Wang* and Tsungnan Lin*†
*Graduate Institute of Communication Engineering
†Department of Electrical Engineering
National Taiwan University, Taipei, 10617 Taiwan
{hpwang, tsungnan}@ntu.edu.tw

*Abstract*—**This paper deliberates on various critical aspects in evaluating searching networks. Existing metrics either draw biased conclusions regarding search performance or provide wrong guidelines for algorithm design. We, therefore, define a unified criterion, *Search Efficiency* (*SE*), to objectively address search performance in a comprehensive manner. The goal of *SE* is to better characterize performance of searching networks than existing metrics do as well as to guide the design of future ones. We first validate the correctness of *SE* in performance evaluation in an ideal graph, strictly binary tree, by analyzing *SE* for two typical search methods, breadth first search and random walk. We further show its strength in performance characterization in the real-world topology, power-law random graph, under various network conditions. We finally design an algorithm, *dynamic search*, based on *SE* analysis. Its proved outstanding performance demonstrates the strength of *SE* to provide guidance for the future design of searching networks.**

*Keywords—performance evaluation, complex networks, search algorithms, peer-to-peer networks*

## I. Introduction

Searching networks, including social networks and computer networks, play an increasingly important role in human activity. A significant example is the recently popular peer-to-peer (P2P) file-sharing systems, e.g. Gnutella and KaZaA, where every peer collaboratively forms a searching network to locate desired files by a real-time search. In the social context of searching networks, people search their acquaintances for a particular item or expertise in a specific domain. Their acquaintances in turn report whether they have the desired item (expertise) or subsequently deliver this query to their next-step acquaintances. In this fashion, a social searching network or so called *human acquaintanceship graph* [8] is formed. Thus, a searching network is a system where each participant contributes to the network and collaborates to help others search targeted resources.

In a searching network, one of the critical issues is to maximize search performance by choosing or designing algorithms used to perform the search process. Novel algorithms [5, 6, 7] have been proposed to address different search aspects, such as success rate, search cost, coverage, or number of hits, but an objective and comprehensive evaluation metric is missing. As a result, these algorithms tend to be designed with biased considerations and evaluated in limited dimensions.

Breadth-first search (BFS) and random walk (RW) [5] are two basic and typical search methods in searching networks. BFS inherently maximizes the search speed and coverage but risks generating search queries in an uncontrolled (exponential) manner. RW, on the other hand, minimizes search cost but generates limited search coverage and results. As a result, one might draw distinct conclusions about algorithm performance, if different metrics are concerned. For example, Gkantsidis et al. [12] claimed RW performs better than BFS in terms of number of hits and failure probability give the same search cost for BFS and RW, but implicitly assumed an infinite search time for RW, which is clearly unfair. Jiang et al. [9] evaluated their proposed search scheme only by search coverage and message cost, leaving search speed and success rate unchecked. Lv et al. [5] provided a spectrum of aspects on evaluation, but analyzed them individually and still lacked an overall consideration.

Our work, therefore, deals with these one-sided perspectives and synthesizes a unified search criterion, *Search Efficiency* (Section II), which is critical particularly in P2P endeavors, to objectively evaluate search algorithms and provide overall guidance for the design of searching networks.

With the unified metric *SE*, we first validate its correctness by deriving its mathematic formulas for BFS and RW in a simple topology, strictly binary tree (SBT), and analyzing whether the performance indicated by *SE* is reasonable. Furthermore, we extend the results of Newman [1] and Adamic [2] and further consider "redundancy" to analytically approximate *SE* for BFS, M-BFS [14], and RW in a power-law random graph (PLRG), which is shown to be the real topology of current searching networks. We thus validate *SE* in comparison with previous simulation works [5, 9, 11], deliver the unique performance characterization of *SE*, and provide in-depth analysis.

Throughout the analysis in this paper, we compare various existing metrics with *SE* to address their limitation and strength. We show that no matter in SBT or PLRG, existing metrics draw biased conclusions regarding search performance; they either provide one-sided considerations or deliver wrong guidelines for algorithm design. Moreover, they fail to characterize performance variance under distinct network conditions, such as object replication ratios (Section III) and object distributions (Section VI).

In the final analysis, we propose a new algorithm, *dynamic search*, based on the results of *SE* analysis. We prove this algorithm outperforms existing ones and *SE* effectively provides guidance for algorithm design.

This work was supported in part by Taiwan National Science Council under grant 93-2213-E-002-057, and by Quanta Computer Inc. under grant 092E0048.



0-7803-8968-9/05/$20.00 (C)2005 IEEE

In summary, our contributions are stated as follows:

- We propose a unified and objective metric, *Search Efficiency*, for evaluating searching networks and characterizing search algorithms.
- We mathematically analyze critical performance metrics—search coverage, cost, success rate, number of hits, and *SE*—in searching networks.
- We analytically evaluate various algorithms, including BFS, M-BFS, RW, and a novel search, in SBT and PLRG, under uniform and non-uniform object distributions.
- We devise a new search algorithm, *dynamic search*, based on the knowledge from temporal *SE* analysis. It is shown to outperform other existing ones, thus *SE* proved to provide solid guidance for algorithm design.

The rest of this paper first follows with the definition and explanation of *Search Efficiency* in Section II. We then analytically derive the general forms of *SE* and provide in-depth discussion on the performance of BFS and RW in SBT in Section III and PLRG in Section IV. Section V presents the novel algorithm, *dynamic search*. We analyze algorithms under non-uniform object distribution in Section VI, then finally conclude in Section VII.

## II. Search Efficiency

We argue that to best characterize the efficiency of any system is to measure its ability to transfer its input to generate meaningful output, which is applicable in the evaluation of search methods performed in any network. In a social network, the input of a search largely involves the cost required for querying process including costs of phone calls, transportation, and even consulting. As for output, it should be measured by searchers' satisfaction in terms of the chance of success, the response speed, and quality of responsive results. To clarify the definitions of and relations between these inputs and outputs in the context of searching networks, we start a series of discussions about *Search Efficiency* with *Query Efficiency* (*QE*).

### A. Query Efficiency

In general, the most critical aspects of search performance involve the extent of search coverage (output) [2] and the cost required to cover the network (input) [5]. By search coverage, denoted as *Coverage* or *C*, we mean the number of distinct or *effective* peers visited by search queries, i.e. we do not count the repeatedly visited ones. In addition, by cost, denoted by *QueryMsg*, we mean the number of queries incurred, for it is a representative factor to which other cost factors (e.g. computer processing power or costs for phone calls and transportation) tend to be proportional. Thus it is trivial to say a search which uses *S* query messages to traverse distinct *S* nodes is perfectly or 100% efficient in terms of query generation. Additionally, we can define a sort of efficiency as *Coverage* / *QueryMsg*. However, the end goal of searching is not to cover as many nodes in the network as possible. Rather, its ultimate goal is to search out the desired targets or objects, in which covering is only one of the adequate conditions (e.g. cache or previous experience) for that end. This is true when the searching network is well-designed, e.g. Chord [13], such that large search coverage is not necessary, or when objects are intentionally deployed in which directed search is preferred. We will show performance difference between *Coverage* and *QueryHits* under non-uniform object distribution in Section VI.

Thus, we define *QueryHits*(*t*) as the number of desired objects found "at" search time *t*, which is measured by the number of hops or depths, to quantify the yields of a search. We introduce the factor search time *t* for the purpose of future discussion. Again, we might define the efficiency of queries as $\sum QueryHits(t)/QueryMsg$. However, this definition is sensitive to the population of desired objects, which is irrelevant to the performance of search algorithms themselves and should be factored out. For this purpose, we introduce the notion of object replication ratio *R* defined as the ratio of the number of targeted objects to the network size (*N*). To cancel the population factor out, we normalize it with respect to *R* and thus formulate *Query Efficiency* (*QE*) as

$$Query\ Efficiency(\%) = \frac{\sum_{t=1}^{TTL} QueryHits(t)}{QueryMsg} \times \frac{100\%}{R}, \qquad (1)$$

where *TTL* refers to the termination condition of searches, measured in hops. To exemplify, we suppose a search consuming 100 messages to find 1 targeted object in a network with *R* of 1%, which reveals that 1% of nodes have the desired object. By (1), *QE* = 100% and we thus call it a perfectly query-efficient search. Furthermore, if the objects are uniformly distributed in the network, we can reasonably claim that the search effectively covers 100 nodes (from 1/1% = 100) and this provides a clear view of the perfect efficiency.

### B. Responsiveness

One of the goals of searching, as addressed previously, is to find out possible objects while the other is to find them as soon as possible. We define search response time, denoted by *t*, measured by discrete numbers of hops, to evaluate the speed of searching objects, or *responsiveness* of a search. If a search finds *Q* desired objects in its $h^{th}$ step or in its $h^{th}$-nearest acquaintances, we denote it as *QueryHits*(*t*=*h*)=*Q*.

We argue that a search getting hits in a faster fashion delivers better users' experience and should be gauged as higher reputation. More specifically, *responsiveness* of a search should be inversely proportional to the response time *t*. To consider this factor for *SE*, we may simply divide *QE* by the weighted response time, which is computed by $\sum_t[t \cdot QueryHits(t)] \ / \ \sum QueryHits(t)$. However, this method would generate unjust results. For example, we assume a search that uses 1000 messages to get 99 hits at $t = 1$ and 1 hit at $t = 100$ with $R = 10\%$, resulting in a weighted response time of $(1 \cdot 99 + 100 \cdot 1)/100$ or 1.99. According to *QE* in (1), if we don't count the hit at $t = 100$, the search is 99% query efficient, but it dramatically reduces to 50.25% efficiency due to dividing by response time 1.99 when that hit is calculated. This method unreasonably emphasizes the slow search hit. We argue that any query hits contribute positively to the search

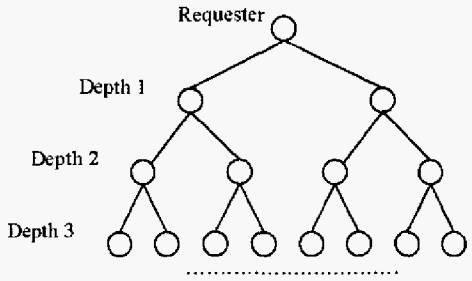


Fig. 1. A strictly binary tree with the requester at the root

itself despite long response time. We thus aggregate these responsive hits rather than divide by the averaged response time to give efficiency as

$$\frac{\sum_{t=1}^{TTL} QueryHits(t)/t}{QueryMsg} \times \frac{100\%}{R}.$$

The efficiency of this example becomes 99.01% rather than 50.25%, where the last found hit contributes 0.01% to efficiency, rather than severely reducing it.

*C. Reliability*

The last concern is reliability, which is measured by *SuccessRate* in our design of *SE*. We introduce it so as to further consider the satisfaction of user experience. Consider two searches (A and B), each performing two runs, as shown in Table I. We assume all objects are found at the same response time. The success rate of Search A is 50% while B is 100%.

TABLE I
SEARCH DATA FOR ILLUSTRATING SUCCESSRATE

| | Search A | | Search B | |
|---|---|---|---|---|
| | *QueryMsg* | *QueryHits* | *QueryMsg* | *QueryHits* |
| Run1 | 100 | 2 | 100 | 1 |
| Run2 | 100 | 0 | 100 | 1 |

Note that if we compute efficiency without *SuccessRate*, we will gain the same result for Search A and B. However, one of the runs in Search A (Run 2) fails and thereby we neglect to measure the penalty of user experience in Run 2. By introducing the term *SuccessRate*, *SE* of Search B remains the same, but *SE* of A is halved. In this manner, it successfully addresses the user satisfaction level while the two searches get the same number of hits at the same message costs. In sum, the term *SuccessRate* is aimed to successfully measure the satisfaction level from users' perspective. Finally, we define the overall criterion for evaluating searching by

$$Search\ Efficiency = \frac{\sum_{t=1}^{TTL} QueryHits(t)/t}{QueryMsg} \times \frac{SuccessRate}{R}, \quad (2)$$

where *TTL* stands for the limit of search covering.

*D. Limitations of Search Efficiency*

The design goal of *SE* is to capture a simple but representative view of search performance. As a result, it is possible to consider more complex considerations for search evaluation. We list three possible aspects that are not covered by *SE*:

1) In the context of computer searching networks, the implementation of caches or DHT would significantly improve the search performance, which *SE* could reflect. However, *SE* doesn't consider the additional resources (processing power or memory) required by performance-boosted mechanisms, such as hash functions or caches, thus potentially overestimating the efficiency of algorithms adopting these additional mechanisms.

2) The costs of searching each computer or peer should not be equally weighted. Consulting an institution for recommendations is clearly more costly than asking a close friend, although we only assume they are equally costly.

3) We make a limited measure of *responsiveness* by the factor $t$. In some applications, such as peer-to-peer telephony, (Skype), search response time is highly concerned while in others not. Therefore, it would be more flexible using $f^{\alpha}$, $\alpha > 0$, to adjust the extent to which search responsiveness is concerned.

By means of *Search Efficiency*, we can objectively evaluate performance of algorithms in searching networks. In the remaining of this paper, therefore, we aim to characterize various existing search algorithms in terms of *SE* and demonstrate the biased view of existing search metrics compared with *SE*. In the following sections, we will mathematically derive the formulas for *SE* in the context of three basic search approaches, BFS, RW and M-BFS, the variation of BFS, in two representative topologies, the strictly binary tree (SBT) as well as the power-law random graph (PLRG), in order to demonstrate the strength of *SE*.

## III. STRICTLY BINARY TREE

We assume an $N$-vertex strictly binary tree whose depth is about $\log_2 N$ and that the requester is at the root such that the response time ($t$) of a query hit is the same as the depth ($d$) where the target object is located. This tree is shown in Fig. 1. Moreover, for simplicity of analysis, we assume objects are *uniformly* distributed in the tree or graph until Section VI.

Before analyzing specific algorithms, we first prepare two common factors for the derivation. Firstly, the number of objects searched out (*QueryHits*) is proportional to the search coverage $C$. Thus, we have

$$QueryHits = R \times C. \quad (3)$$

Secondly, the success rate of a search is also relevant to the search coverage. To begin with, we know that each node owns the target object with a probability of $R$; that is, each node lacks the object with a probability of $1-R$. Suppose a search covers $C$ vertices and thus the probability these $C$ nodes share no targeted object is $(1-R)^C$. Inversely, the probability these $C$ nodes share one or more objects, or equivalently *SuccessRate*, is determined by

$$SuccessRate = 1-(1-R)^C. \quad (4)$$

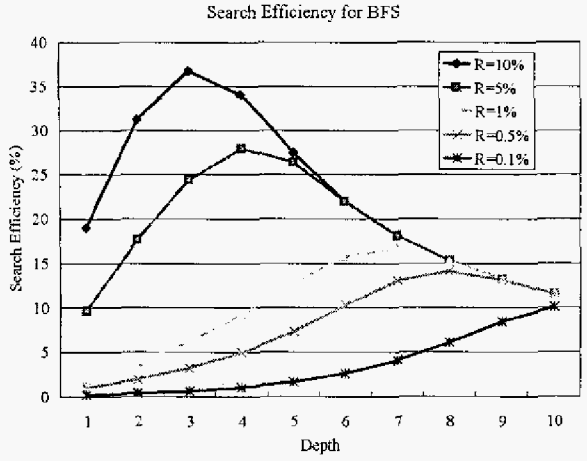


Fig. 2. *Search Efficiency* for BFS terminated by incremental *TTL*s (Depth) in a strictly binary tree with various replication ratios *R*

### *A. Breadth First Search in Strictly Binary Tree (SBT)*

**Analytic Derivation:** Breadth-first search (BFS) performs by broadcasting the received queries to all neighbors except where the received query came from. Therefore, by the regular structure of a strictly binary tree, the search coverage terminated at depth *TTL* is given by

$$Coverage\ (C) = \sum_{t=1}^{TTL} 2^t. \qquad (5)$$

Furthermore, the number of messages required to traverse the tree is the same as the quantity of its search coverage due to the very nature of BFS. Thus, $QueryMsg = C = \sum_t 2^t$. According to (1), (3), and (5), we attain

$$QueryEfficieny|_{BFS} = \frac{R \cdot \sum_{t=1}^{TTL} 2^t}{\sum_{t=1}^{TTL} 2^t} \times \frac{100\%}{R} = 100\%. \qquad (6)$$

Surprisingly, the formula of $QE_{BFS}$ yields a constant, 1 or 100%, regardless of the replication ratio *R* or the termination depth *TTL*. By the definition of *QE*, this means that BFS is a perfectly query-efficient search in the context of a binary tree; that is, BFS generates *no* redundant messages while traversing a binary tree. The idea of *redundancy* will be further defined and discussed in the next section.

Finally, the general formula of *SE* defined in (2) for BFS in a binary tree is

$$SE_{BFS} = \frac{\sum_{t=1}^{TTL} 2^t/t}{\sum_{t=1}^{TTL} 2^t} \times \left[1-(1-R)^{\sum_{t=1}^{TTL} 2^t}\right]. \qquad (7)$$

The derived $SE_{BFS}$ is complex for one to gain insight of its properties due to the running variable *t* and various possible values of *R*. To deliver a clearer understanding, we assume the replication ratio $R << 1$, which is true in real searching networks, and approximate (7) as

$$SE_{BFS} \cong \frac{\sum_{t=1}^{TTL} 2^t/t}{\sum_{t=1}^{TTL} 2^t} \times \left[1-\left(1-R\cdot\sum_{t=1}^{TTL} 2^t\right)\right] = R \cdot \sum_{t=1}^{TTL} 2^t/t \ . \qquad (8)$$

**Search Efficiency Analysis:** To exemplify $SE_{BFS}$, we set $R = 0.1\%$ (far less than 1) and obtain by (8) $SE_{TTL=1} = 0.2\%$, $SE_{TTL=2} = 0.4\%$, and $SE_{TTL=3} = 0.67\%$. Note *SE* is strictly increasing with respect to *TTL*—$SE_{TTL=2}$ is exactly twice of $SE_{TTL=1}$ and $SE_{TTL=3}$ is more than three times of $SE_{TTL=1}$. The reasons are two-fold. Firstly, as formula (6) shows, BFS in a binary tree is perfectly query-efficient, which means every query positively contributes to its search coverage and in turn produces promising increase in *SE*. Secondly, the speed at which query hits are returned is faster than the decay factor of response time *t*. Furthermore, formula (8) tells that the benefits from BFS are increasingly proportionally to $2^t$ while the factor *t* is used to compensate the demerit of long search time, where the factor $2^t$ tends to dominate. Thus we conclude every query or every additional covered depth makes a positive contribution to the overall performance despite the compensation of time, given that the replication ratio is much smaller than unity.

We present analytically-derived data of $SE_{BFS}$, without approximation, by (7) with a spectrum of parameters, *R*s and *TTL*s, in Fig. 2. Firstly, we note that $SE_{BFS}$ for all *R*s approaches some fixed level in the long run. This fixed level, obtained by (7) for large *t*, is determined by the characteristic of the searched topology—strictly binary tree—that is irrelevant to *R*. Second, the short-term increase of *SE* for high *R* (10% or 5%) results from the perfect query efficiency and popularly distributed objects, while the long-term decrease is due to the compensator of response time *t*. If we use the notion $t^{\alpha}$ suggested in Section II.*D*, where α is 0 or small for some application scenarios and responsiveness is of little concern, *SE* in (7) will increasingly grow to some fixed level. Third, as for low *R* (0.1% or 0.5%), the results in Fig. 2 are reflective of the discussions in the above paragraph—*SE* is consistently increasing.

Note that, however, if we take *TTL* as infinity in (7), it gives zero seemingly contradicting our notion. In reality, however, *TTL* cannot be infinity but is generally 7~10, in which *SE* still generates a fixed level of performance reflecting the characteristics of SBT.

**Metrics Analysis:** We compare two metrics, *SE* and *Coverage* in this scenario. The results of *Coverage* of BFS can be referred to in Fig. 3(c). If we take only *Coverage* (*C*) into consideration, it produces the same performance in spite of different extents of object replication (different values of *R*) since *C* by (5) is independent of *R*. Hence, *Coverage* fails to characterize the performance variance in searching networks with different replication ratios. On top of this, if the design goal is to maximize *C*, then one may conclude that the choice of termination condition *TTL* is the larger the better—an impractical conclusion. On the other hand, if we only inspect *QueryMsg*, we will get entirely opposite conclusions. Therefore, *Coverage* and *QueryMsg* draw contradictory conclusions and fail to provide comprehensive guidance.

In fact, by the indication of *SE* in Fig. 2, *TTL* should be small when *R* is large in order to avoid unnecessary message propagation when *R* is large and to generate satisfactory results when *R* is small. In sum, *SE* better characterizes performance and provides a better guideline of *TTL* design than *Coverage* and *QueryMsg*.

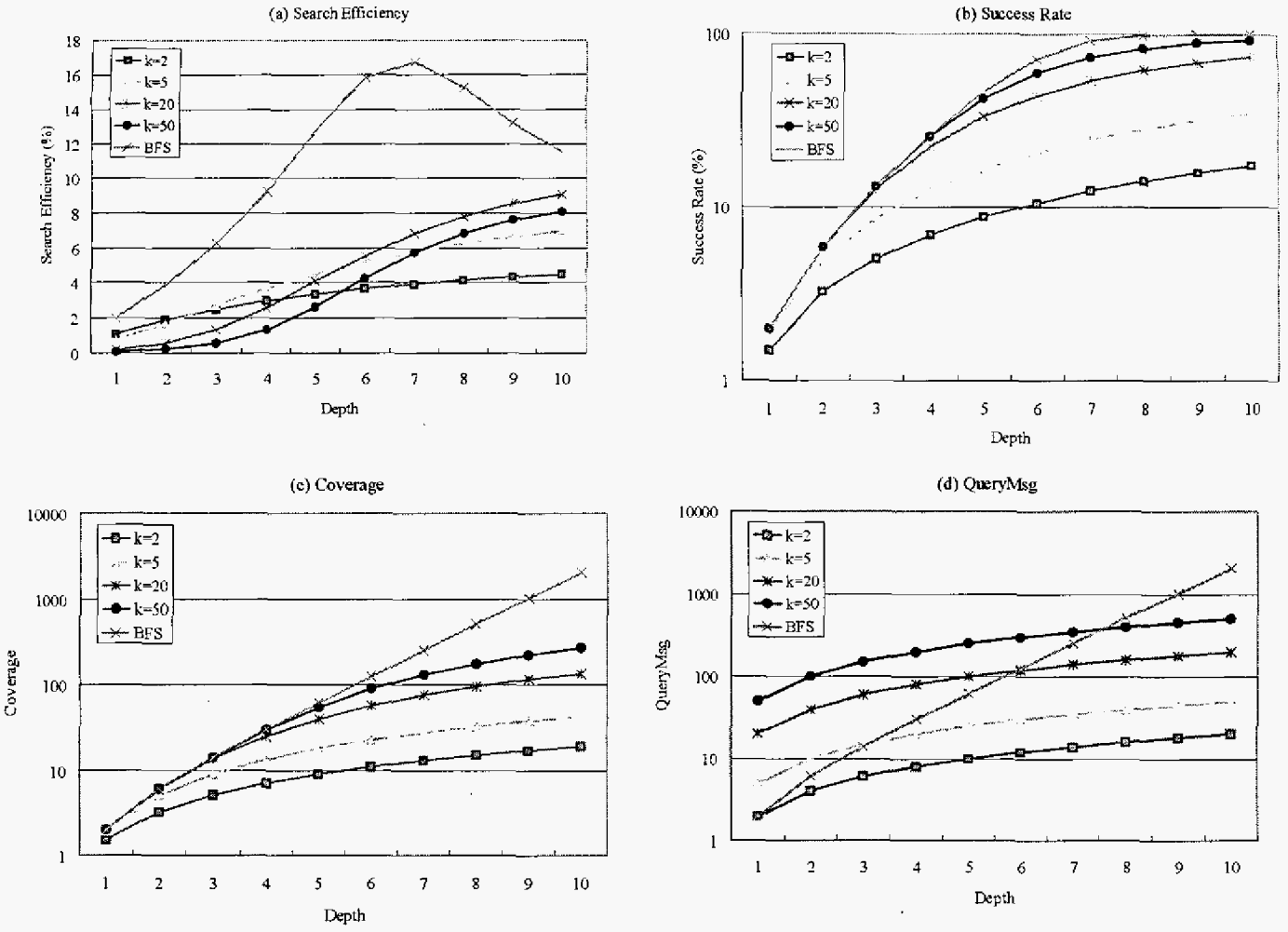


Fig. 3. Performance comparison by various metrics—(a) *Search Efficiency*, (b) *SuccessRate*, (c) *Coverage*, and (d) *QueryMsg*—for RW of various number of walkers $k$ and for BFS in a strictly binary tree with $R = 1\%$

*B. Multiple Random Walks in Strictly Binary Tree*

When it comes to RW search, we use multiple "walkers" to traverse the network and the number of walkers is denoted by $k$. Each walker independently searches the network and randomly chooses one of the next-hop neighbors to continue its journey to the limit of *TTL* hops.

**Analytic Derivation:** To begin with, we consider *Coverage* to derive *SE*. We know each vertex at depth $t$ is visited by a random walker with equal probability, $1/2^t$. Moreover, each random walker *independently* makes its own decisions to traverse the topology. Thus, the probability that all $k$ walkers don't visit a certain vertex is $(1-1/2^t)^k$. As a result, at depth $t$, the average number of nodes visited (*Coverage per Depth*) by $k$ random walkers is given by the expectation

$$E(X)_t = 2^t\left[1-(1-1/2^t)^k\right]. \quad (9)$$

By (3), $QueryHits(t) = R \cdot E(X)_t$. Moreover, the query messages of random walk are generated per hop for each walker until terminated by the *TTL* limit, hence

$$QueryMsg = k \cdot TTL. \quad (10)$$

As a result, *QE* of $k$-random walk is

$$QE|_{RW=k} = \frac{\sum_{t=1}^{TTL} R \cdot E(X)_t}{k \cdot TTL \cdot R} = \frac{\sum_{t=1}^{TTL} E(X)_t}{k \cdot TTL}. \quad (11)$$

Furthermore, from (4), we obtain

$$SuccessRate = 1-(1-R)^C = 1-(1-R)^{\sum_{t=1}^{TTL} E(X)_t}. \quad (12)$$

Therefore, *Search Efficiency* for $k$-random walks is

$$SE|_{RW=k} = \frac{\sum_{t=1}^{TTL} E(X)_t / t}{k \times TTL} \times \left[1-\left(1-R\right)^{\sum_{t=1}^{TTL} E(X)_t}\right], \quad (13)$$

where $E(X)_t$ is determined by (9).

**Search Efficiency Analysis:** Assuming $R$ = 1%, we generate a series of performance results of *SE* in terms of various numbers of walkers $k$. We thus plot these results of *SE* (13), *SuccessRate* (12), *Coverage* (9), and *QueryMsg* (10) for RW and BFS in Fig. 3.

In Fig. 3(a), we observe that all *SE*s of RW consistently increase with respect to the depth or search time. Nevertheless, they all are smaller than that of BFS due to too many (redundant) query messages in the local search, and the slow covering and low *SuccessRate* in the long-term search. Therefore, they fail to utilize the regular structure of SBT. As for the number of walkers $k$, a too large (e.g. 50) or too small (e.g. 2) value of $k$ gives degraded performance, thus resulting

in strong sensitivity in the choice of $k$.

**Metrics Analysis:** By merely inspecting Fig. 3(b) for *SuccessRate* or (c) for *Coverage*, one may jump to a conclusion that the number of walkers $k$ is the larger the better. This aspect disregards the fact that larger $k$ would generate larger search cost, shown in Fig. 3(d), and potentially redundant query messages. In fact, comparing RW of $k = 50$ and of $k = 20$, we find that their values of *SuccessRate* or *Coverage* during depth $t = 1$~4 are almost the same while the former generates 2.5 times more search cost—the latter search uses less search cost to produce similar search fruits. In consequence, in the short-term search, the latter one should be gauged as better search. Thus the conclusion larger $k$ is better for RW would be fallacious. Therefore, we argue that neither *SuccessRate* nor *Coverage* is a good performance indicator.

Moreover, the long-term performance will inherent the short-term so that *SE* in Fig. 3(a) well characterizes the better performance for RW of $k = 20$. Besides, RW of $k = 2$ would be the best search in Fig. 3 if we try to minimize *QueryMsg* and scalability is the most concerned issue. Yet, this would be a specious conjecture since it entirely flies in the face of the final end of search—to find the results responsively.

### C. Summary of Search Efficiency in SBT

By the discussion in this section, we validate *SE* by showing 1) the 100% $QE_{BFS}$ indicates that BFS perfectly utilizes the regular structure of SBT and generates no redundant messages, 2) the sagging $SE_{RW}$ reveals RW fails to take advantage of the structure of SBT, and 3) the fixed level of $SE_{BFS}$ in long-term search effectively reflects the characteristics of SBT. The first two results can be confirmed by intuition and thus verify the correctness of *SE*. The third observation further demonstrates the superiority of *SE* in characterizing search performance under specific topologies.

Through metrics analysis, we have demonstrated that existing metrics, *Coverage*, *QueryMsg*, and *SuccessRate*, are one-sided and may lead to biased conclusions. They cannot distinguish performance variance in searching networks when replication ratios are distinct, and cannot provide reasonable guidance in the design of parameters *TTL* and $k$ while *SE* can.

## IV. Power-law Random Graph

In a random graph or a realistic network, its topology is not structurally organized but formed in an ad-hoc manner. Adamic's work [3] demonstrated that the current Internet follows a power-law degree distribution where a few web pages or web sites are extremely highly-connected while others are weakly linked. In a power-low random graph (PLRG), the probability a vertex has degree $k$ is $p_k$ proportional to $k^{-\tau}$ where $\tau > 0$. In this and the following sections, we will use PLRG as the network topology to explore the efficiency of various search algorithms.

### A. Review of Generating functions

To mathematically describe a power-law random graph, we use the generating function formalism introduced by Newman et al. [1] with arbitrary degree distributions. We first let $G_0(x)$ be the generating function for the distribution of the vertex degree $k$ in a random graph. Then

$$G_0(x) = \sum_{k=0}^{\infty} p_k x^k \tag{14}$$

where $p_k$ is the probability that a randomly chosen vertex on the graph has degree $k$.

For a graph with a power-law distribution with exponent $\tau$, minimum degree $k = 1$ and an abrupt cutoff at $m = k_{max}$, the generating function is then given by

$$G_0(x) = c \sum_{k=1}^{m} k^{-\tau} x^k$$

with $c$ a normalization constant, depending on $m$ and $\tau$ to satisfy the normalization requirement $G_0(1) = 1$.

The average degree of a randomly chosen vertex is given by

$$z_1 = \langle k \rangle = \sum_{x=1}^{m} k \cdot p_k = G'_0(1) \tag{15}$$

Another important quantity is the distribution of the degree of the vertex which we arrive at by following a randomly chosen edge. Such an edge arrives at a vertex with probability proportional to *the degree of that vertex*, and the vertex thus has a probability distribution of degree proportional to $kp_k$. By [1], the distribution of outgoing edges (except the one we have come from) of that vertex, one of the first or immediate neighbors, is generated by the function

$$G_1(x) = \frac{G'_0(x)}{G'_0(1)} = \frac{1}{z_1} G'_0(x).$$

The generating function for probability distribution of the number of *second-nearest* neighbors of the original vertex can be written as $G_0(G_1(x))$ in the limit of large $N$ ($N$ is the network size). Hence, the average number $z_2$ of *second neighbors* is

$$z_2 = \left[ \frac{d}{dx} G_0(G_1(x)) \right]_{x=1} = G'_0(1) G'_1(1) \tag{16}$$

Furthermore, the work in [1] generalizes (16) so that the average number $z_h$ of the $h^{th}$ nearest neighbors is

$$z_h = \left[ G'_1(1) \right]^{h-1} G'_0(1). \tag{17}$$

Besides, according to approximation in [2], we have

$$G'_0(1) \cong \frac{1}{\tau - 2} \left( 1 - m^{2-\tau} \right), \tag{18}$$

and

$$G'_1(1) \cong \frac{1}{G'_0(1)} \frac{m^{3-\tau}}{(3-\tau)}, \tag{19}$$

assuming $2 < \tau < 3$.

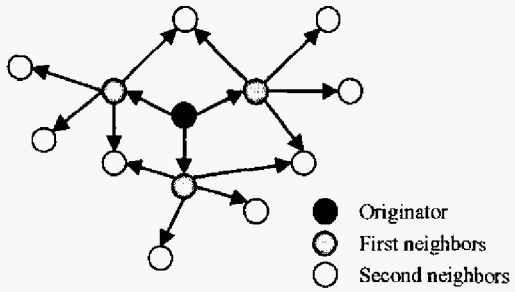


Fig. 4. A random graph for illustrating "redundancy"

*B. Redundancy in Power-law Random Graph*

Equation (17) equivalently tells us that the average number of the $h^{th}$ neighbors is strictly the product of the average degree of each vertex, $G'_0(1)$, and the average outgoing degree of vertices arrived by a randomly chosen edge, $G'_1(1)$, to the $(h\text{-}1)^{th}$ power, given the graph size $N$ is infinity. However, in reality—when $N$ is not infinite—it is simply not the case specified in (17) where the number of $h^{th}$ neighbors is geometrically increasing. In other words, $z_h$ should not be geometrically increasing due to the "redundancy" in random graphs. By *redundancy* we mean edges of any vertex that leads to repeatedly visited vertices, resulting in a fewer effective number of vertices reached by edges than the number of traversed edges. Thus, to express in the terms of search networks, we use a definition similar to [10]:

*"A search network N has 'redundancy' if there exists a link (edge) in N that can be removed without reducing any vertex's search coverage, which is generated by certain search algorithm."*

To quantify the redundancy of a graph by certain search algorithm, we define "redundancy" as

$$Redundancy = 1 - \frac{No.\ of\ Vertices\ Effectively\ Reached}{No.\ of\ Edges\ Ever\ Traversed} \quad (20)$$

Note that redundancy may actually be useful to improve the fault tolerance of the system, since if one peer fails, another can perform its processing. Moreover, redundancy may be useful to reduce response time if a peer stands at a redundant edge closer to the searcher. Thus, fault tolerance and search latency tradeoff with efficiency when redundancy is concerned.

We illustrate this notion of redundancy by Fig. 4, in which we draw a graph with 13 vertices and 15 edges where the black node is the search originator, gray nodes are the first neighbors of the originator, and the white nodes are the second neighbors. Arrows show the directions and paths of message forwarding by BFS. Inspecting this graph, we have the number of first neighbors of the black node, $G'_0(1) = 3$, and the degree of outgoing edges of each first neighbor, $G'_1(1) = 4$. Nevertheless, the effective number of second neighbors is 9, not simply the product of $G'_0(1)$ and $G'_1(1)$, 12, as specified by (16). Thus, we obtain the redundancy by (20) as 1 – (3+9) / (3+12), or 1/5, which means in this case one-fifth (20%) of the edges are redundant by a BFS search.

*C. Breadth First Search in Power-law Random Graph*

**Analytic Derivation:** To analytically quantify the redundancy of a random graph, we first derive the number of *second* neighbors $z_2$ covered by BFS, which it is ideally $G'_0(1)G'_1(1)$ by (16). However, according to the discussion in Section V.B, $z_2$ will be lower than the ideal value when $N$ is not infinite due to the graph redundancy. To derive $z_2$, it is largely equivalent to solve the problem that what the number of balls (vertices) ever chosen (or inversely left not chosen) is when choosing $G'_1(1)$ balls out of $N$ balls and put them back, and repeat this procedure $G'_0(1)$ times, with $G'_1(1) < N$. For simplicity, we first assume the probability each ball (vertex) to be chosen is uniform. Thus the probability that each ball is un-selected is $1 - [G'_1(1)/N]$ after one time of this procedure. After $G'_0(1)$ times of the procedure, the probability each ball selected becomes

$$1-\left[1-G'_1(1)/N\right]^{G'_0(1)}.$$

Hence, if we assume all balls are chosen uniformly and the expectation of the effective number of chosen balls (second neighbors) is

$$\sum_{i=1}^{N} 1-\left[1-G'_1(1)/N\right]^{G'_0(1)} = N\left\{\ 1-\left[1-G'_1(1)/N\right]^{G'_0(1)}\right\}.$$

when neglecting the chance to repeatedly reach the first neighbors.

However, vertices are arrived at by edges with probabilities proportional to their degrees [1], rather than uniformly, as previously stated (Section V.A). Suppose the probability each vertex to be reached by certain edge is $p_i$ for i = 1, 2, ..., $N$ and $p_1 + p_2 + \ldots + p_N = 1$. In a power-law random graph, the probability $p_i$ of vertex $i$ is proportional to its degree and equivalently given by

$$p_i \propto \frac{m}{i^{1/\tau}} = \frac{m/i^{1/\tau}}{\sum_{i=1}^{N} m/i^{1/\tau}}, \quad (21)$$

such that $\sum_i p_i = 1$, where $m$, the maximum degree, is set by $N^{1/\tau}$ [4].

If ignoring the chance to revisit the first neighbors, we could approximate the effective number of second neighbors as

$$\sum_{i=1}^{N}\left\{\ 1-\left[1-p_i\cdot G'_1(1)\right]^{G'_0(1)}\right\},$$

where we assume $p_i << 1$ as $G'_1(1) << N$, which is true in general cases. Note that the term $p_i \cdot G'_1(1)$ approximately represents the expectation of vertex $i$ to be visited with $G'_1(1)$ independent selections, each of which only selects one vertex among the $N$ ones (with returning back). This term is surely not the exact expectation of second neighbors (the actual value should be a little smaller), but an approximation, which holds when $p_i \cdot G'_1(1)$ is much smaller than unity.

To generalize it, the effective number of vertices arrived at the $h^{th}$ depth or hop (*Coverage per Depth* or $C_h$) for $h \geq 2$ could be approximated by

$$C_h = z_h = \sum_{i=1}^{N}\left\{1-\left[1-p_i\cdot G'_1(1)\right]^{z_{h-1}}\right\}.$$

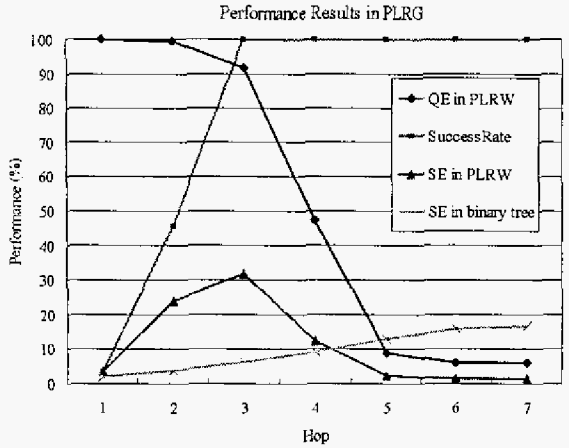


Fig. 5. Performance results in percentage of *QE*, *SuccessRate*, and *SE* in PLRG and *SE* in a strictly binary tree for BFS with $R$ = 1%

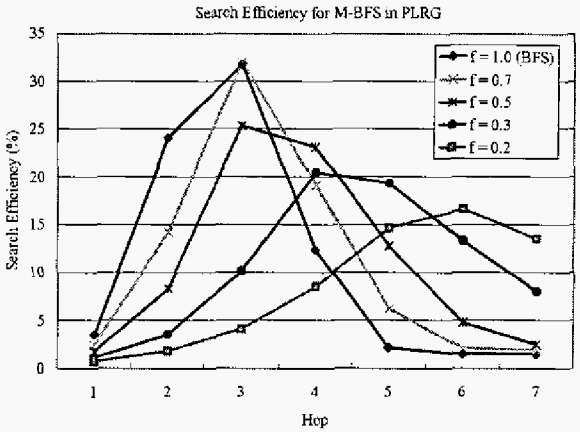


Fig. 6. *Search Efficiency* for M-BFS of various fraction parameters $f$ in a power-law random graph with $R$ = 1%

Nonetheless, this formula doesn't consider the possibility the search revisits previously reached vertices, which is significant when the search is in the deeper depth. To eliminate this problem, we first let $V_h$ be the event that a vertex is visited at the $h^{\text{th}}$ depth or hop, then the probability vertex $i$ is visited at the $h^{\text{th}}$ hop is

$$P_i(V_h) = \begin{cases} p_i \cdot G'_0(1), \text{ for } h = 1 \\ 1 - \left[1 - p_i \cdot G'_1(1)\right]^{C_{h-1}}, \text{ for } h \geq 2. \end{cases} \quad (22)$$

Therefore, the average number of non-repeatedly visited vertices at the $h^{\text{th}}$ hop by BFS in PLRG is

$$C_h = z_h = \begin{cases} \sum_{i=1}^{N} P_i(V_h), \text{ for } h = 1 \\ \sum_{i=1}^{N} \prod_{j=1}^{h-1} \left[1 - P_i(V_j)\right] \cdot P_i(V_h) \text{ for } h \geq 2, \end{cases} \quad (23)$$

where $P_i(V_h)$ is given by (22). Intuitively, *Coverage* or $C$ is given by $\sum_h C_h$.

To derive the redundancy, we let the number of edges traversed or equivalently the number of queries generated at the $h^{\text{th}}$ hop (depth) be $e_h$. That is, *QueryMsg* is given by

$$e_h = \begin{cases} G'_0(1), \text{ for } h = 1 \\ G'_1(1) \cdot z_{h-1}, \text{ for } h \geq 2. \end{cases} \quad (24)$$

Thus, the redundancy of BFS terminated at $h = TTL$ is determined by

$$Redundancy|_{BFS} = 1 - \frac{\sum_{h=1}^{TTL} z_h}{\sum_{h=1}^{TTL} e_h}. \quad (25)$$

Note that $QE_{BFS}$ is equivalent to 1−*Redundancy*. Furthermore, according to (2), (3), (4), and (25), we obtain

$$SE|_{BFS} = \frac{\sum_{h=1}^{TTL} C_h / h}{\sum_{h=1}^{TTL} e_h} \times [1 - (1-R)^{\sum_{h=1}^{TTL} C_h}] \times 100\%, \quad (26)$$

where $C_h$ is specified by (23) and $e_h$ by (24).

A variation of BFS is Modified-BFS (M-BFS) [14], which adopts a fraction parameter $f$ to serve as the probability that each search agent uses to forward the query message to its neighbors. For example, if $f$ = 0.5 and certain search agent has 10 neighbors, then it will forward the received query message to 0.5·10 or 5 of its neighbors (randomly). Since its operation is similar to BFS, the formula of *SE* specified by (26) still holds for M-BFS, where $C_h$ is given by (23), except

$$P_i(V_{h\geq 2}) = 1 - \left[1 - f \cdot p_i \cdot G'_1(1)\right]^{C_{h-1}}, \quad (27)$$

$$P_i(V_{h=1}) = f \cdot p_i \cdot G'_0(1), \quad (28)$$

and $e_{h\geq 2} = f \cdot G'_1(1) \cdot C_{h-1}$ and $e_{h=1} = f \cdot G'_0(1)$.

**Performance Analysis:** We use the following parameters throughout this paper for the power-law network: $N$ = **10,000**, exponent $\tau$ = **2.1**, $R$ = **1%**, and $m = N^{1/\tau} \sim$ **80**. These parameters are similar to those used in [2]. By (18) and (19), $G'_0(1)$ = 3.55 and $G'_1(1)$ = 16.21. Therefore, we present the performance results, through a series of calculations of (21), (23), (25), and (26), in Fig. 5.

Note that *QE* in Fig. 5 is not as perfect as that in a binary tree, but decays dramatically during $h$ = 3~5, where the redundancy comes from the exponentially generated messages, which approves the results in [5] and [9]. Furthermore, *SE* in PLRG is significantly high compared with that in a strictly binary tree with the same $R$ in the short-term search, while *SE* in the tree is superior to that in PLRG in the long term. In sum, BFS performs better in the local and inefficiently in the global, when deployed in the power-law random graph. Similar conclusion is drawn in the work [11] by its simulation results.

For M-BFS, we generate the data of *SE* of various fraction parameters with the same settings used for BFS ($N$ = 10,000 and $\tau$ = 2.1) and plot them in Fig. 6. We find that the fraction parameter controls the extent to which performance increases in the local or decreases in the global—the larger the parameter $f$ is the more greatly the performance changes. Hence, if the search is aimed to gain great performance increase in the short term, we should take larger $f$; on the other hand, smaller $f$s give relatively consistent *SE* by compromising the fast performance increase in the short-term search. Therefore, the choice of the fraction parameter depends on whether the short-term satisfaction or long-term efficiency is more concerned.

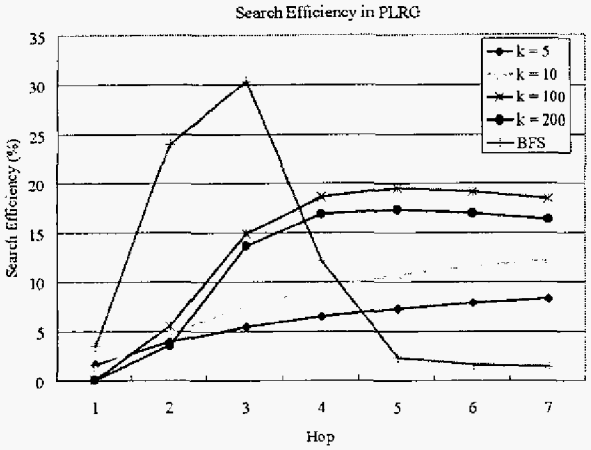


Fig. 7. *Search Efficiency* for RW of various number of walkers $k$ and for BFS in a power-law random graph with $R$ = 1%

*D. Multiple Random Walks in PLRG*

The property of random walk is dramatically different from BFS. The former traverses a graph in a random and unpredictable fashion while the latter operates rather regularly. In particular, the concept of *depth* used in BFS is not applicable in random walk in that the walkers may go "back and forth" in the graph so that we could only describe them with respect to *hop* rather than depth. Therefore, we represent search coverage in terms of *Coverage per Hop* ($C_h$).

**Analytic Derivation:** To derive the analytic formulas of performance metrics, we first obtain the number of "candidates" that RW might traverse at the $h$ hop, which is conceptually similar to the number of $h^{th}$ neighbors of BFS, $z_h$, except RW doesn't have the concept of "depth." We denote that for RW as $r_h$. Let $R_h$ be the event a vertex is the candidate of RW at hop $h$ (in the $h^{th}$ neighbors of RW), then the probability vertex $i$ is the candidate of RW at hop $h$ is

$$P_i(R_h) = \begin{cases} p_i \cdot G'_0(1), & \text{for } h = 1 \\ 1-\left[1-p_i \cdot G'_1(1)\right]^{C_{h-1}}, & \text{for } h \geq 2. \end{cases} \quad (29)$$

Then, the average number of candidates of RW at hop $h$ is

$$r_h = \sum_{i=1}^{N} P_i(R_h), \quad (30)$$

where $P_i(R_h)$ is given by (29).

Since random walkers have the behavior similar to those in the binary tree if the forwarding candidates are known, we apply the line of reasoning in the binary tree for PLRG. Hence, the probability vertex $i$ is visited at hop $h$ for RW is

$$P_i(V_h) = P_i(V_h \cap R_h) = P_i(R_h) \cdot P_i(V_h | R_h) = P_i(R_h) \cdot \left[1-\left(1-1/r_h\right)^k\right]. \quad (31)$$

To deal with the phenomenon vertices may be revisited, we apply the line of reasoning in BFS in PLRG. Therefore, the formula of $C_h$ in (23) still holds for RW except using $P_i(V_h)$ of RW (31). Thus, $SE$ for random walk with $k$ walkers is given by

$$SE|_{RW=k} = \frac{\sum_{h=1}^{TTL} C_h}{k \cdot TTL} \times \frac{1-(1-R)^{\sum_{h=1}^{TTL} C_h}}{R},$$

where $C_h$ specified is by (23), in which $P_i(V_h)$ is formulated by (31).

**Search Efficiency Analysis:** By the same conditions for BFS, we plot $SE$ for RW of various numbers of walkers $k$ and re-plot $SE$ for BFS for comparison in Fig. 7. This figure shows RW generates consistently increasing performance in most cases of $k$, which can be answered by its controlled fashion of message generation and granular coverage that have been suggested in [5]. In addition, the curve of $k$ = 2,000 reasonably explains the redundancy generated by too many walkers despite its fine properties in PLRG. Inspecting the curve of BFS, it outperforms RW in the local search but inversely in the global ($h \geq 5$), which confirm the simulation results in [11].

In sum, $SE$ well characterizes the delayed performance increase of RW and its consistent long-term performance.

*E. Summary of Search Efficiency in PLRG*

Based on the unified metric $SE$ and its temporal analysis, we better characterize that, in PLRG, BFS gains its excellent performance in the local search space but decays rapidly in the long-term search, M-BFS controls its performance increase or decrease by the fraction parameter $f$, and RW performs consistently in the global search space while its performance increase in relatively slow in the short-term search. Furthermore, by the analysis of $SE$ and $QE$, we can explain the causes behind the ostensive phenomena: the great short-term performance of BFS stems from its aggressive search to deliver responsive results while keeping little redundancy in the local and the long-term performance suffers from the overwhelming search cost generated while it still retrieves satisfactory results in the global. On the other hand, the delayed performance increase of RW is due to its conservative search and redundancy in the local while its conservatism trades for relatively little redundancy and thus consistent performance in the global.

In particular, our work for PLRG strongly reflects previous works (simulations) in various respects and in turn is validated for its ability of characterizing, especially in terms of temporal analysis. Besides, $SE$ analysis indicates the choice of the fraction parameter depends on whether the short-term satisfaction or long-term efficiency is more concerned.

Thus far, we have shown the potency of $SE$ in performance characterizing and reasoning. We will further demonstrate its strength in guiding the design of search algorithms by inventing a new search based on $SE$ and validate the performance improvement of the new search, in the following section.

## V. Dynamic Search: An Algorithm Designed Based on Search Efficiecy

Evaluation metrics are critical in judging search performance. If *Coverage* is the only metric concerned, one may conclude that BFS is the best search algorithm despite the overwhelming search cost. It overlooks the system load and the aspect of operation efficiency. Moreover, if search cost is the most important criterion of a searching network, RW would be the best appropriate algorithm for that system. However, it fails to evaluate the ability to achieve the final end of searching networks—to search out targeted results responsively. In consequence, biased metrics may draw biased conclusions and provide wrong guidelines for system design. Thus, we endeavor to devise a new search based on the comprehensive metric, *SE*, in order to demonstrate the strength of *SE*. In addition to its strength in performance characterization and reasoning, we show the strength of *SE* to serve as the design guidance of the invented algorithm—*dynamic search*.

We attempt to utilize the merits of the three analyzed algorithms from the viewpoint of *SE* for the new search. Accordingly, on the basis of the conclusions drawn in Section IV.E, the new algorithm should resemble BFS in short-term searches, mimic RW for long-term propagation, and be able to fine tune the performance through certain parameters as used in M-BFS. Therefore, we separate the search process into two phases. In the threshold phase (local space), the search is similar to BFS with some dynamic tuning forwarding probabilities; in the ultimate phase (long-term space), it operates as the random walk search to consistently retain the performance gained from the threshold phase. The detailed operations are described in the following subsection.

### A. Operation

Dynamic search starts as a probabilistic search with dynamic fraction parameter $f_h$ at different hops $h$ when $h \le n$. For $h > n$, it switches to the random walk search. In the threshold phase, it operates as M-BFS but with dynamic $f_h$, for $h = 1, 2, \ldots, n$. For example, for dynamic search with $n = 2$, $f_1 = 1$, and $f_2 = 0.5$, the search agents at $h = 1$ perform BFS, perform M-BFS with $f = 0.5$ at $h = 2$ and operate as random walk for $h \ge 3$. Moreover, in the random-walk phase, the number of walkers $k$ is determined by the outstanding query messages or the effective search agents covered at the $h^{th}$ hop, that is, $C_h$.

Hence, the behavior of dynamic search changes dynamically in terms of time (hop) to adapt to the appropriate search properties in different phases. Hopefully, in terms of *SE*, it would outperform other algorithms in each phase thanks to the fine-tuned design.

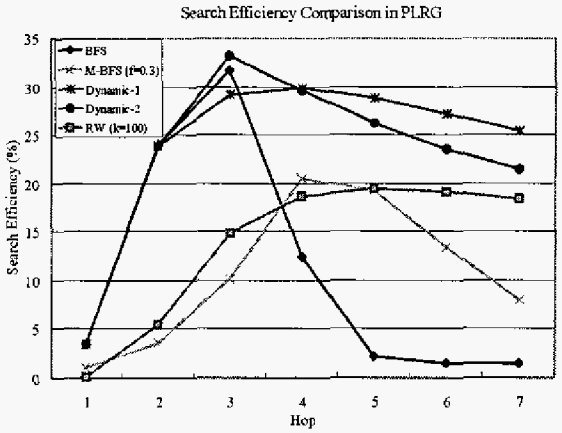


Fig. 8. *Search Efficiency* comparison for various algorithms: BFS, M-BFS ($f$=0.3), RW ($k$=100), and the dynamic search in PLRG with $R$ = 1%.

TABLE II
Parameter Design for Dynamic Search in Fig 8

| | $n$ | $f_1$ | $f_2$ | $f_3$ |
|---|---|---|---|---|
| Dynamic-1 | 2 | 1.0 | 1.0 | N/A |
| Dynamic-2 | 3 | 1.0 | 1.0 | 0.3 |

### B. Performance Analysis

To analyze the characteristics of dynamic search, we use the knowledge we have learned in previous sections where we mathematically formulate *SE*. In this section, we analyze only in the PLRG. The general form of *SE* in (26) applies for dynamic search and $C_h$ is given by (23), except $e_{h=1} = f_1 \cdot G'_0(1)$, $e_{2 \le h \le n} = f_h \cdot G'_1(1) \cdot C_{h-1}$, $e_{h>n} = C_{h=n}$, and

$$P_i(V_h) = \begin{cases} f_h \cdot p_i \cdot G'_0(1), \text{ for } h = 1 \\ 1 - \left[1 - f_h \cdot p_i \cdot G'_1(1)\right]^{C_{h-1}}, \text{ for } 2 \le h \le n \\ P_i(R_h) \cdot \left[1 - \left(1 - 1/r_h\right)^k\right], \text{ for } h > n, \end{cases} \quad (32)$$

where $r_h$ is specified by (30).

As for the parameter design, we refer to the observation in Fig. 6, where BFS performs the best in the first two hops and lower $f$s for M-BFS achieve more consistent performance in the long-term search. Thus, we design two sets of parameters: the first one, Dynamic-1, performs BFS in the first two hops and random walks in the following phase ($n$ = 2); the second one, Dynamic-2, performs BFS in the first two hops, M-BFS with $f$ = 0.3 at the third hop, and then random walks ($n$ = 3). The number of walkers $k$ is in RW is dynamically determined by the number of outstanding queries at hop $n$, i.e. $C_{h=n}$. The detailed parameters are shown in Table II.

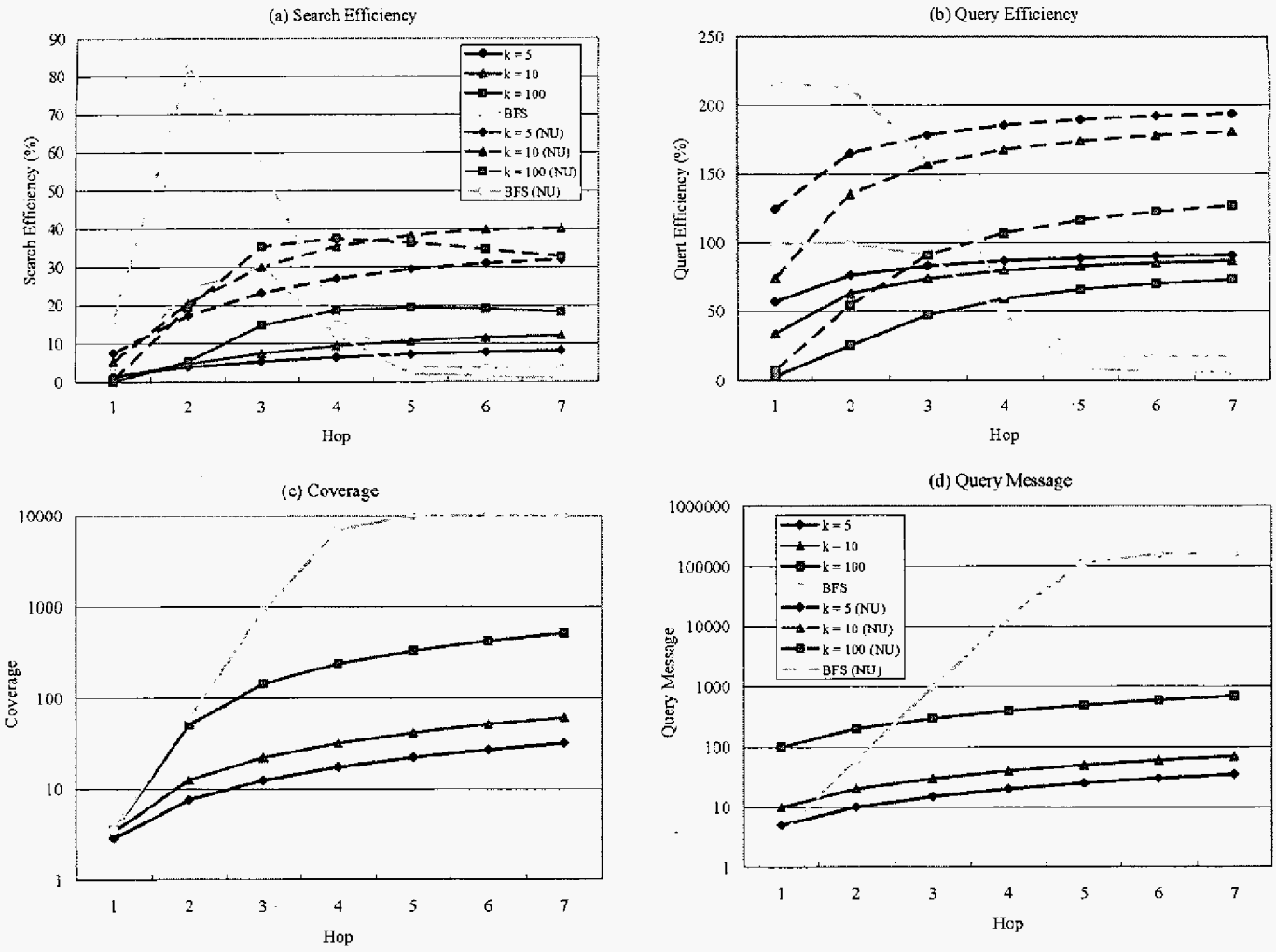


Fig. 9. Performance comparison by various metrics—(a) *Search Efficiency*, (b) *Query Efficiency*, (c) *Coverage*, and (d) *QueryMsg*—for RW of various number of walkers $k$ and for BFS in PLRG with $R = 1\%$ under uniform and non-uniform (NU) object distribution. Solid lines represent data of uniform distribution and dashed-lines represent non-uniform distribution.

We generate *SE* of Dynamic-1 and -2 and make performance comparison with BFS, M-BFS ($f = 0.3$), and RW ($k = 100$) in Fig. 8. We take M-BFS with $f = 0.3$ in order to compare with Dynamic-2, which uses $f_3 = 0.3$. And we use 100 as the number of walks for RW since it generates the best performance (in Fig. 7).

In Fig. 8, we can observe that dynamic searches outperform other algorithms especially in the long-term search. They resemble BFS within $h \leq 2$ as expected and perform consistently as random walk does, thus outperforming others in long-term search as we design. Note that Dynamic-2 trades its performance at $h = 3$ for its long-term efficiency by using a low probability $f = 0.3$, and vice versa for Dynamic-1.

## VI. Non-Uniform Object Distribution

Throughout our analysis, for simplicity we had assumed the object distribution as uniform. However, this assumption leads to the conclusion that *QueryHits* equals to $R \cdot Coverage$, which violates our argument in Section II.A that *Coverage* is only one of the conditions to produce *QueryHits*. To support our argument and justify our consideration of *QueryHits* in *SE* rather than *Coverage*, we analyze *SE* under a non-uniform object distribution as proposed in [11].

In this object distribution, the probability a search agent (vertex) owns certain object is proportional to its degree $d$. Let $O$ be the event that certain search agent owns the targeted object, then the probability agent $i$ has the object is determined by

$$P_i(O) \propto d_i = \frac{R \cdot N \cdot d_i}{\sum_{j=1}^{N} d_j}, \tag{33}$$

such that $\sum_i P_i(O) = R \cdot N$, where $d_i = m / i^{1/\alpha}$ [4].

**Analytic Derivation:** Since the object distribution is not uniform, we cannot simply use $R \cdot Coverage$ to represent *QueryHits*, which in fact is formulated by

$$QueryHits(h) = \begin{cases} \sum_{i=1}^{N} P_i(O) \cdot P_i(V_h), & \text{for } h = 1 \\ \sum_{i=1}^{N} \prod_{j=1}^{h-1} \left[1 - P_i(O) P_i(V_j)\right] \cdot P_i(O) P_i(V_h), & \text{for } h \geq 2, \end{cases} \tag{34}$$

where $P_i(V_h)$ is given by (22) and $P_i(O)$ by (33).

For *SuccessRate*, we generalize the form $1-(1-R)^C$ in (4) for the uniform distribution to deliver the one in non-uniform distribution:

$$SuccessRate = 1 - \prod_{i=1}^{N} \prod_{j=1}^{h} \left[ 1 - P_i(O) P_i(V_j) \right]. \quad (35)$$

Now, equations (34), (35), and (24) suffice to solve *SE* defined by (2) for BFS.

For RW, *QueryHits*($h$) follows formula (34) derived in BFS and *SuccessRate* follows (35) in BFS, where $P_i(V_h)$ is given by (31) and $P_i(O)$ by (33).

**Search Efficiency Analysis:** We plot analytic data in Fig. 9, where the dashed-lines represent the data under non-uniform (NU) object distribution. We use the same colors to represent searches with identical parameters. We find that *SE* in Fig. 9(a) is significantly increased under NU distribution for both BFS and RW. The performance increase is around 75% ~ 250% for RW at $h$ = 7 and 250% for BFS at $h$ = 2. This can be explained by the graph property that vertices tend to connect to those with higher degrees [1, 2], which has been validated by simulations in [11].

**Metrics Analysis:** Fig. 9(c) indicates that every search in question generates identical *Coverage* under different object distributions, and Fig. 9(d) draws the same conclusion for *QueryMsg*. Therefore, these two metrics totally fail to distinguish the performance variance under NU distribution. Moreover, in Fig. 9(b), *Query Efficiency*, defined by *QueryHits*/(*QueryMsg*·$R$), explains the performance increase by indicating more *QueryHits* found given that same number of *QueryMsg*. In consequence, *SE*, in which *QE* is a critical element, well characterizes the performance difference in the two scenarios.

## VII. Conclusion

This paper defines a unified metric, *Search Efficiency* (*SE*), addressing performance in searching networks in terms of *Query Efficiency*, responsiveness, and reliability. Mathematical formulas and approximations of *SE* and other existing metrics are derived to characterize performance and provide in-depth analysis for various search algorithms. We justify the correctness of *SE* in performance evaluation by analyzing it in an ideal topology, strictly binary tree. We further demonstrate its ability to characterize search performance in a large-scale PLRG, the real-world network topology.

We conclude that existing metrics either leads to biased conclusions regarding performance or fail to reflect performance variance when network conditions change. Moreover, they tend to provide wrong guidelines for the design of various algorithm parameters (e.g. *TTL*, $k$, and $f$). The proposed metric, *SE*, effectively characterizes the performance variance under different network conditions and delivers objective and in-depth performance analysis.

In the final analysis, the outstanding performance of *dynamic search*, the new algorithm devised based on the guidance of *SE*, manifests the efficacy of *SE* to conduct design of search algorithms. Therefore, our proposal of *SE* contributes to providing guidance for the future design of searching networks.


## Acknowledgement

We thank James Donald for the helpful discussion.


## References


[1] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distribution and their applications," *Phys. Rev. E*, 64:026118, 2001.

[2] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in power-law networks," *Phys. Rev. E*, 64:046135, 2001.

[3] L. A. Adamic, "The small world web," *Proceedings of the 3rd European Conf. on Digital Libraries*, volume 1696 of Lecture notes in Computer Science, pages 443-452. Springer, 1999.

[4] W. Aiello, F. Chung, and L. Lu, "A random graph model for massive graphs," *Proceedings of the thirty-second annual ACM symposium on Theory of Computing*, pages 171-180, 2000.

[5] Q. Lv, P. Cao, E. Cohen, K. Li and S. Shenker, "Search and replication in unstructured peer-to-peer networks," *Proc. of ACM ICS*, June 2002.

[6] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," *Proc. of IEEE ICDCS*, July 2002.

[7] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks," Technical Report CS-TR-4451, University of Maryland, 2003.

[8] S. Milgram, "The small-world problem," *Psychology Today*, 1:62-67, 1967.

[9] S. Jiang, L. Guo and X. Zhang, "LightFlood: an Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems," *Proc. of IEEE ICPP*, Oct. 2003.

[10] B. F. Cooper and H. Garcia-Molina, "SIL: Modeling and measuring scalable peer-to-peer search networks," *Proc. of International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Berlin, 2003.

[11] T. Lin, H. Wang, and J. Wang, "Search Performance Analysis and Robust Search Algorithm in Unstructured Peer-to-Peer Networks," *Proc. of IEEE CCGrid*, April 2004.

[12] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," *Proc. of IEEE INFOCOM*, March 2004.

[13] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *Proc. of ACM SIGCOMM*, 2001.

[14] V. Kalogeraki, D. Gunopulos and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," *Proc. of ACM CIKM*, Nov. 2002.