LETTER *IEICE/IEEE Joint Special Section on Autonomous Decentralized Systems*

Hop-Value-Based Query-Packet Forwarding for Pure P2P[**]

Masato UCHIDA[†*a)] ***and*** **Shinya NOGAMI**[†], ***Members***

SUMMARY In pure peer-to-peer (P2P) file sharing applications and protocols using a flooding-based query algorithm, a large number of control packets (query packets) are transmitted on the network to search for target files. This clearly leads to a degradation of communication quality on the network and terminals as the number of users of the application increases. To solve such problems, this paper proposes: (1) a unified framework to describe a wide variety of query algorithms for pure P2P and (2) a new query algorithm based on this framework. Our framework determines the number of destinations for query packets based on the hop value recorded in received query packets. Simulation results revealed that the proposed query algorithm can reduce the overhead in the flooding-based query algorithm and *k*-random walks without decreasing the success rate of retrieval regardless of the density of target files in the network.

key words: *P2P, query algorithm*

1. Introduction

The past few years have seen the development of several new file sharing applications and protocols based on the pure peer-to-peer (P2P) model, which does not have any servers to support clients. In this model, every client acts as a server. These so-called "servents" (servent = server + client) form a decentralized and unstructured application-level overlay network on the physical layer, by connecting to existing servents. Several pure P2P applications pass messages (packets) that implement file sharing among servents on the overlay network. For example, query packets for target files are broadcast on the overlay network. This flooding-based query-packet-forwarding algorithm (flooding-based query algorithm) is clearly not scalable because it will obviously lead to an overhead, such as an overwhelming amount of query traffic and a high CPU load, as more servents join the overlay network. Against this background, the possibility of multiple random walks (*k*-random walks) on a pure P2P network to find target files has been discussed [1]. In the *k*-random walks, a requesting node sends *k* query packets, and each query packet takes its own random walk on the network. Although another search algorithm has been studied [2], it needs an additional function to cache query results that cannot be obtained by the usual pure P2P protocols. However, there may be better query algorithms than flooding and *k*-random walks that are based on the usual pure P2P protocols. Therefore, we first provide a unified framework that can describe a wide variety of query algorithms. Based on this framework, we then propose a new query algorithm that can reduce the overhead without decreasing the success rate of retrieval regardless of the density of the target files in the network.

Our framework includes the flooding-based query algorithm because it can determine the number of destinations for query packets based on the hop value recorded in received query packets. Note that the hop value is information that the query packet usually has. We evaluated various query algorithms within the framework (flooding, *k*-random walk, proposed) by simulation, where the algorithms ran on a model drawn from actual topology data [3].

2. Related Work

2.1 Flooding-Based Query Algorithm

Several pure P2P applications, such as Gnutella and its clones, pass query packets that implement file sharing among servents on the overlay network. Specifically, each servent forwards the received query packets to all of its neighbors. Each packet's header contains a time-to-live (TTL) field. TTL is used in the same fashion as in the IP protocol: at each hop its value is decremented until it reaches zero, at which point the packet is discarded. That is, TTL is the number of times the packet will be forwarded by servents before it is removed from the network. Each packet's header also contains a hop field. The hop value is incremented at each hop. That is, the hop value is the number of times the packet has been forwarded.

2.2 *k*-Random Walks

Since the flooding-based query algorithm cannot be scaled because each retrieval clearly generates a large amount of query traffic, the so-called "*k*-random walks" was proposed as an alternative [1]. In the *k*-random walks, the requesting node sends *k* query packets, and each query packet takes its own random walk on the network.

Manuscript received March 25, 2005.
Manuscript revised June 23, 2005.

[†]The authors are with NTT Service Integration Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.

[*]Presently, with Network Design Research Center, Kyushu Institute of Technology.

[**]An earlier version of this paper was presented at the 18th International Workshop on Communications Quality & Reliability (CQR 2004).

a) E-mail: m.uchida@ndrc.kyutech.ac.jp

DOI: 10.1093/ietcom/e88–b.12.4517

3. Unified Framework for Query Algorithms

3.1 Proposed Framework

In this section, we first provide a unified framework that allows us to describe a wide variety of query algorithms. Then, we give examples of how algorithms are set up within the framework.

The key idea of the proposed framework is to utilize the hop value h recorded in the received query packets when they are forwarded. That is, a servent forwards the received query packet to some of its neighbors according to h, while a servent that uses the flooding-based query algorithm forwards the received query packet to all of its neighbors regardless of h.

The proposed framework is outlined in Fig. 1, where the crossed circle is connected with the circled circle, closed circles, and open circles. When the crossed circle receives a query packet (squared square) whose TTL is t and hop value is h from the circled circle, it performs the following procedure, where the nodal degree of the crossed circle is $n + 1$.

Step 1: If $t = 0$, the crossed circle drops the query packet; otherwise, it determines the number of destinations $N(n, h)$.

Step 2: If $N(n, h) \leq n$, the crossed circle selects $N(n, h)$ servents (closed circles) randomly from n servents without multiplicity (Fig. 1, left); otherwise it selects $N(n, h)$ servents (closed circles) randomly from the n servents with multiplicity (Fig. 1, right), excluding the servent that transmitted the incoming query packet (circled circle).

Step 3: The crossed circle decrements t and increments h of the received query packet. Then, it forwards the query packets (crossed squares) to the selected servents.

We can construct various query algorithms based on the above by setting $N(n, h)$ appropriately. For example, the definition enables us to describe both the flooding-based query algorithm and k-random walks within the framework by setting $N(n, h)$ as follows.

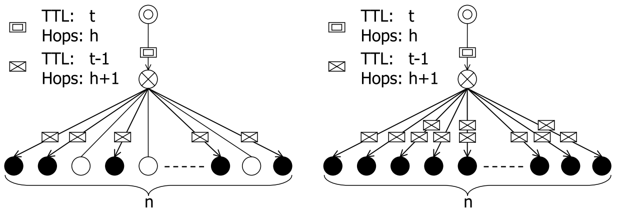

Fig. 1 Query packet forwarding algorithm for $N(n, h) \leq n$ (left) and $N(n, h) \geq n$ (right). (circle: servent, square: query packet)

$$N_1(n,h) = \begin{cases} n & \text{for } h \leq d_1 \\ 0 & \text{for } h > d_1 \end{cases}, \quad (d_1\text{: constant}),$$

$$N_2(n,h) = \begin{cases} k & \text{for } h \leq d_2 \\ 1 & \text{for } h > d_2 \end{cases}, \quad (d_2, k\text{: constant}).$$

Obviously, $N_1(n, h)$ is equivalent to the flooding-based query algorithm, and $N_2(n, h)$ is equivalent to the k-random walks when $d_2 = 0$. This means that the proposed framework is a natural extension of the notion of query algorithms.

3.2 Proposed Query Algorithm

In this section, we propose a new query algorithm based on the proposed framework. The following three conditions are required to construct a better query algorithm than flooding and k-random walks: high success rate of retrieval (Condition 1) and low overhead (Conditions 2 and 3), where $m \geq n$ and $g \geq h$.

Condition 1: For the same hop value, the larger the nodal degree is, the larger the number of query packet destinations should be.

$$N(m, h) \geq N(n, h) \geq 0$$

Condition 2: For the same nodal degree, the larger the hop value is, the smaller the number of query packet destinations should be, because if the hop value is large, a lot of servents have already received query packets.

$$0 \leq N(n, g) \leq N(n, h)$$

Condition 3: The larger the nodal degree is, the more the number of query packet destinations should be decreased, because if the nodal degree is large, query packets are more likely to be forwarded to servents that have already received ones.

$$0 \leq N(n, h) - N(n, g) \leq N(m, h) - N(m, g)$$

Note that the flooding-based query algorithm (i.e., $N_1(n, h)$) satisfies only Condition 1 and k-random walks (i.e., $N_2(n, h)$) satisfies only Condition 2.

Now, we propose a new query algorithm based on the above framework. It is defined as follows.

$$N_3(n,h) = \begin{cases} n & \text{for } h \leq d_3 \\ \lceil n^{\frac{1}{1+h-d_3}} \rceil & \text{for } h > d_3 \end{cases}, \quad (d_3\text{: constant}).$$

In $N_3(n, h)$, the number of destinations for query packets that will be forwarded decreases as the hop value grows depending on the nodal degree. We can easily confirm that Conditions 1, 2, and 3 are satisfied by $N_3(n, h)$. Although there are many examples similar to $N_3(n, h)$, we found heuristically that $N_3(n, h)$ gives better performance than the examples we tried in simulations. Finding the best form of $N(n, h)$ theoretically is still an open problem.

4. Simulation

In this section, we consider the appropriate forms of $N(n,h)$ that can reduce the number of query packets without decreasing the success rate of retrieval. To achieve this, we compared the performance of $N_1(n,h)$, $N_2(n,h)$, and $N_3(n,h)$ through simulation. Though $N(n,h)$ can take other values, we can obtain a rough approximation by evaluating these forwarding algorithms.

4.1 Preparation

We performed the simulation on the proposed framework as follows. Here, we refer to the "crawl number 5 network [3]–[6][†]" as the target P2P network for simulation, which is an unstructured power-law random graph with about 2300 servents, where the set of servents forming the network is described by C. The topology of a crawl number 5 network is illustrated in Fig. 2. This "crawl network" is a snapshot of a small portion of a real P2P network that contains servents in which a kind of Gnutella software is installed.

Step 1: Place a target file, which will be searched for, on all servents included in C with probability (density) p, where the number of arrangement patterns of target files is described by NumPlace. That is, we used various arrangement patterns of target files to evaluate the performance of query algorithms because the performance depends on the position of the target files. Note that the mean value of the number of target files in C is $|C| \times p$, where the mean value is calculated over NumPlace arrangement patterns of target files.

Step 2: For each arrangement patterns of target files, randomly select from C a servent that will search for the target file, where the number of selections is NumQuery.

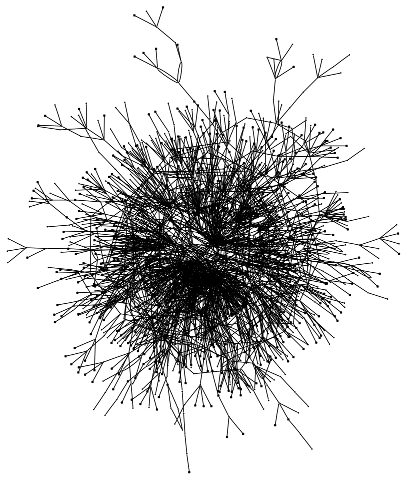

Fig. 2 Network topology of crawl number 5.

Step 3: Transmit the query packets generated by each selected servent, based on the proposed framework in Sect. 3.1.

Note that, for each value of probability p, statistics can be collected NumPlace $\times$ NumQuery times through the above simulation. In this paper, we collected the following statistics concerning both success rate of retrieval and overhead of query algorithm (cf. Fig. 3), where $i = 1, \ldots, \mathrm{NumPlace} \times \mathrm{NumQuery}$.

$s_p(i)$: A value describing whether or not target files will be found before the search terminates (i.e., the value is 1 when the target files are found at least once and 0 otherwise).

$g_p(i)$: Number of (all) generated query packets that each node in the network must process before the search terminates (i.e., the average number of (all) existing query packets in the network before the search terminates).

$v_p(i)$: Number of visited nodes before the search terminates.

$d_p(i)$: Number of duplicated query packets that each node in the network must process before the search terminates, where the number of duplicated query packets is defined as $g_p(i) - v_p(i)$.

Using these statistics, we can define the following criteria, where $|C|$ is the number of elements included in the set C. We evaluated the simulation results for probability p using these criteria.

S_p: Probability of finding the target object before the search terminates.

$$S_p = \frac{\sum_{i=1}^{\mathrm{NumPlace}*\mathrm{NumQuery}} s_p(i)}{\mathrm{NumPlace} * \mathrm{NumQuery}}$$

G_p: Overhead of an algorithm measured by the average number of generated query packets that each node in the network must process.

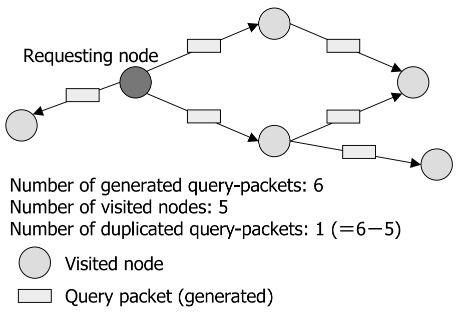
Requesting node
Number of generated query-packets: 6
Number of visited nodes: 5
Number of duplicated query-packets: 1 (=6−5)
Visited node
Query packet (generated)

Fig. 3 Example of evaluation. (flooding-based query-algorithm)

[†]The original network data had been available in [4]. Though the data had been deleted from the site already (see [5]), we can get the same data in [3]. Collection method of the data is given in [6].

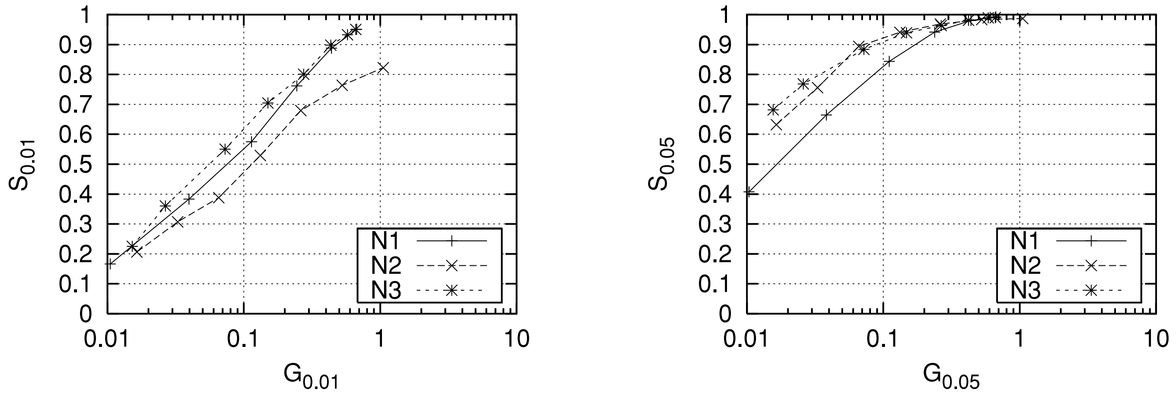

Fig. 4 Success rate of retrieval (S_p) versus the average number of generated query packets (G_p) for $N_1(n,h)$, $N_2(n,h)$, and $N_3(n,h)$. The horizontal axis is a log scale. (Left: $p = 0.01$, Right: $p = 0.05$)

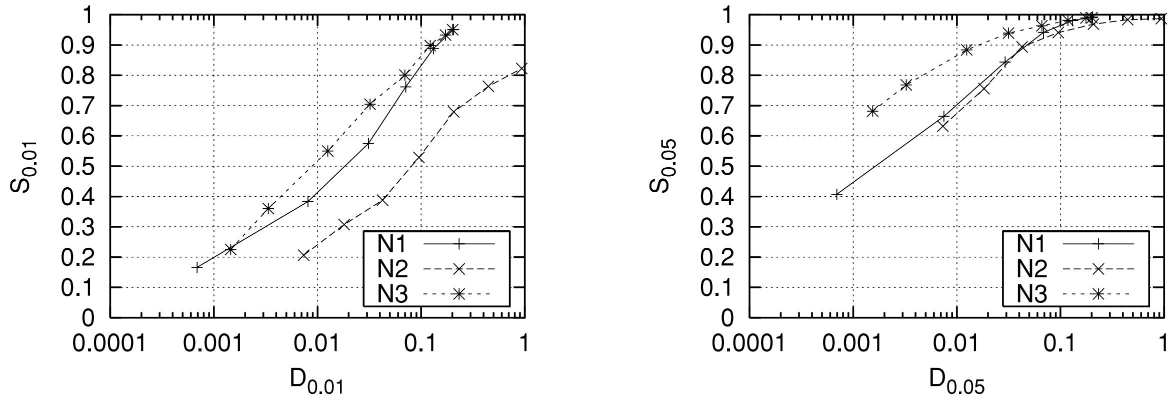

Fig. 5 Success rate of retrieval (S_p) versus the average number of duplicated query packets (D_p) for $N_1(n,h)$, $N_2(n,h)$, and $N_3(n,h)$. The horizontal axis is a log scale. (Left: $p = 0.01$, Right: $p = 0.05$)

$$G_p = \frac{1}{|C|}\frac{\sum_{i=1}^{\text{NumPlace}*\text{NumQuery}} g_p(i)}{\text{NumPlace} * \text{NumQuery}}$$

D_p: Overhead of an algorithm measured by the average number of duplicated query packets that each node in the network must process.

$$D_p = \frac{1}{|C|}\frac{\sum_{i=1}^{\text{NumPlace}*\text{NumQuery}} d_p(i)}{\text{NumPlace} * \text{NumQuery}}$$

4.2 Results and Discussion

The simulation results are shown in Figs. 4 and 5, where the parameter values used for these simulations are listed in Table 1. Here, these parameter values are valid for comparing the performance of $N_1(n,h)$, $N_2(n,h)$ and $N_3(n,h)$. This is because we can confirm the difference between the properties of these query algorithms in Figs. 4 and 5 using these values. A detailed discussion about the difference between the properties is given in this section. The simulation results are summarized in Table 2. Although these figures are for crawl5, the results were almost the same even when other topology data was used.

We measured the overheads of algorithms by G_p and D_p. Note that the average number of generated query packets per node (G_p) and the average number of duplicated query packets per node (D_p) increased as the values of d_1, k, and d_3 rose. The detailed relationship between the values of parameters (d_1, k, and d_3) and the values of criteria (G_p, D_p, and S_p) is given in Tables 3, 4, and 5.

Table 1 Parameters values.

NumPlace	20
NumQuery	200
TTL	7
p	0.01, 0.05
d_1	1, 2, 3, 4, 5, 6
d_2	0
d_3	0, 1, 2, 3, 4, 5, 6, 7
k	10, 20, 40, 80, 160, 320, 640

Table 2 Results of simulations.

		N_1	N_2	N_3
G_p	$p = 0.01$	good	bad	good
	$p = 0.05$	bad	good	good
D_p	$p = 0.01$	good	bad	good
	$p = 0.05$	bad	bad	good

As we can see from Fig. 4 (right), which is for $p = 0.05$, $N_2(n,h)$ and $N_3(n,h)$ found target files with higher

Table 3 Relationship between d_1, G_p, and D_p.

d_1	1	2	3	4	5	6
$G_{0.01}$	0.011	0.040	0.11	0.24	0.44	0.67
$D_{0.01}$	0.00069	0.0081	0.031	0.071	0.13	0.21
$S_{0.01}$	0.17	0.38	0.57	0.76	0.89	0.95
$G_{0.05}$	0.010	0.038	0.11	0.24	0.44	0.67
$D_{0.05}$	0.00069	0.0075	0.029	0.069	0.13	0.20
$S_{0.05}$	0.41	0.66	0.84	0.94	0.98	0.99

Table 4 Relationship between k, G_p, and D_p.

k	10	20	40	80	160	320	640
$G_{0.01}$	0.016	0.033	0.66	0.13	0.26	0.53	1.1
$D_{0.01}$	0.0073	0.018	0.042	0.095	0.21	0.45	0.94
$S_{0.01}$	0.21	0.31	0.39	0.53	0.68	0.76	0.82
$G_{0.05}$	0.016	0.033	0.66	0.13	0.26	0.53	0.11
$D_{0.05}$	0.0073	0.018	0.043	0.095	0.21	0.45	0.94
$S_{0.05}$	0.63	0.76	0.89	0.94	0.97	0.98	0.99

Table 5 Relationship between d_3, G_p, and D_p.

d_3	0	1	2	3	4	5	6	7
$G_{0.01}$	0.015	0.027	0.073	0.15	0.27	0.43	0.58	0.66
$D_{0.01}$	0.0015	0.0034	0.013	0.032	0.070	0.12	0.17	0.20
$S_{0.01}$	0.23	0.36	0.55	0.70	0.80	0.90	0.93	0.95
$G_{0.05}$	0.16	0.26	0.72	0.15	0.27	0.42	0.59	0.66
$D_{0.05}$	0.0015	0.0032	0.012	0.032	0.066	0.12	0.18	0.20
$S_{0.05}$	0.68	0.77	0.88	0.94	0.96	0.98	0.99	0.99

probability than $N_1(n,h)$, when $N_1(n,h)$, $N_2(n,h)$, and $N_3(n,h)$ used the same number of generated query packets. Additionally, we can see from Fig. 4 (left), which is for $p = 0.01$, that $N_1(n,h)$ and $N_3(n,h)$ found the target files with higher probability than $N_2(n,h)$, when $N_1(n,h)$, $N_2(n,h)$, and $N_3(n,h)$ used the same number of generated query packets. This means that $N_3(n,h)$ found the target files with a smaller number of generated query packets regardless of the value of p, although $N_1(n,h)$ found target files with a smaller number of generated query packets only when p was low, and $N_2(n,h)$ found target files with a smaller number of generated query packets only when p was high.

Moreover, as we can see from Fig. 5 (right), which is for $p = 0.05$, $N_3(n,h)$ found target files with higher probability than $N_1(n,h)$ and $N_2(n,h)$, when $N_1(n,h)$, $N_2(n,h)$, and $N_3(n,h)$ used the same number of duplicated query packets. Additionally, as we can see from Fig. 5 (left), which is for $p = 0.01$, $N_1(n,h)$ and $N_3(n,h)$ found the target files with higher probability than $N_2(n,h)$, when $N_1(n,h)$, $N_2(n,h)$, and $N_3(n,h)$ used the same number of duplicated query packets. This means that $N_3(n,h)$ found target files with a smaller number of duplicated query packets regardless of the value of p, although $N_1(n,h)$ found the target files with a smaller number of generated query packets only when p was low, and $N_2(n,h)$ found target files with a larger number of duplicated query packets regardless of the value of p. This is because if the query algorithm based on $N_2(n,h)$ is used, the requesting servent's neighbours must transmit numerous redundant query packets when k is high. We therefore conclude that $N_3(n,h)$ can find target files with fewer query packets without duplication, regardless of the value of p, although $N_1(n,h)$ can find target files with fewer query packets without duplication only when p is low, and $N_2(n,h)$ can find target files with fewer query packets by duplication only when p is high.

Finally, a closer look at the qualitative difference between $N_2(n,h)$ and $N_3(n,h)$ in Figs. 4 and 5 for $p = 0.05$ is worthwhile. That is, both $N_2(n,h)$ and $N_3(n,h)$ find target files with almost the same probability when they used the same number of generated query packets (see Fig. 4 (right)), while the number of duplicated query packets used in $N_3(n,h)$ is smaller than that in $N_2(n,h)$. This means that these two query algorithms find target files with almost the same probability, though the number of visited nodes in $N_3(n,h)$ is larger than that in $N_2(n,h)$. This point seems to be paradoxical, because it is natural to say that the probability of finding a target file increases as the number of visited nodes increases. We discuss the seeming paradox in the Appendix. Note that the seeming paradox does not influence the result of this paper summarised in Table 2 if we can resolve the paradox, because the result is valid whether or not we notice the paradox.

5. Conclusion and Further Study

This paper provided a unified framework for query algorithms. This framework decides the number of destinations of query packets based on the hop value, and it can describe the flooding-based query algorithm and k-random walks. We simulated various query algorithms within the framework using actual topology data. As a result, we found a better query algorithm than the flooding-based query algorithm and k-random walks. We conclude that the number of servents to which the query packet will be forwarded should be small as the hop value increases like $N_3(n,h)$. This is because, $N_3(n,h)$ can find target files with higher probability and lower overhead regardless of the value of p, though $N_1(n,h)$ (i.e., flooding-based query algorithm) and $N_2(n,h)$ (i.e., k-random walks) can find target files with higher probability and lower overhead only when the value of p is low or high, respectively.

References

[1] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," Proc. 16th Annual ACM International Conference on Supercomputing, pp.84–95, 2002.

[2] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability," http://www-2.cs.cmu.edu/kunwadee/research/p2p/gnutella.html

[3] Crawl number 5 network, http://cantor.ee.ucla.edu/˜boykin/data/crawl5.log

[4] LimeWire, http://crawler.limewire.org/data.html

[5] Internet Archive, http://web.archive.org/web/*/http://crawler.limewire.org/data.html

[6] DELIS (Dynamically Evolving Large-scale Information Systems) Project, "Measurement methodology for wireless ad hoc multihop networks," http://delis.upb.de/deliverables/D2.2.1.pdf

Appendix: Detailed Relationship between $N_2(n,h)$ and $N_3(n,h)$

Note that the definition of $N_2(n,h)$ and $N_3(n,h)$ given in Sect. 3 indicates that the number of destination in N_2 does not depend on the fluctuation of nodal degree while that in N_3 does. This means that the number of visited nodes in N_3 fluctuates more widely than that in N_2. Therefore, the variance of the number of visited nodes, $v_p(i) = g_p(i) - d_p(i)$, in N_3 is larger than that in N_2.

On the other hand, the theoretical expression of the probability of finding a target file, S_p, is given as

$$S_p = \frac{\sum_{i=1}^{\text{NumPlace}*\text{NumQuery}} \{1-(1-p)^{v_p(i)}\}}{\text{NumPlace} * \text{NumQuery}}. \qquad (\text{A}\cdot 1)$$

In the following, we abbreviate NumPlace as NP and NumQuery as NQ. Moreover, let us define V_p as $G_p - D_p$. Note that $|C|\times V_p$ is the mean value of the number of visited nodes in C.

Executing the Taylor expansion of $1-(1-p)^{v_p(i)}$ for $v_p(i)$ at $|C|\times V_p$, we get

$$\begin{aligned} &1-(1-p)^{v_p(i)} \\ &= 1-\sum_{n=0}^{\infty}\frac{1}{n!}(1-p)^{|C|\times V_p} \\ &\quad \times\{\ln(1-p)\}^n\{v_p(i)-|C|\times V_p\}^n \\ &\approx 1-\sum_{n=0}^{2}\frac{1}{n!}(1-p)^{|C|\times V_p} \\ &\quad \times\{\ln(1-p)\}^n\{v_p(i)-|C|\times V_p\}^n. \end{aligned} \qquad (\text{A}\cdot 2)$$

Here, we ignore the terms containing $\ln(1-p)^n$ for $n \geq 3$. This is because if $p \ll 1$ then $|\ln(1-p)| \ll 1$. Moreover, substituting Eq. (A·2) into Eq. (A·1), we get

$$\begin{aligned} S_p \approx\ & 1-(1-p)^{|C|\times V_p}\Bigg[1+\frac{\{\ln(1-p)\}^2}{2} \\ & \times\frac{1}{\text{NP}\times\text{NQ}}\sum_{i=1}^{\text{NP}\times\text{NQ}}\{v_p(i)-|C|\times V_p\}^2\Bigg]. \end{aligned} \qquad (\text{A}\cdot 3)$$

Finally, defining the mean value and variance of $v_p(i)$ by $\mathbb{E}[v_p]$ and $\mathbb{V}[v_p]$, respectively, we can rewrite Eq. (A·3) as

$$S_p \approx 1-(1-p)^{\mathbb{E}[v_p]}\left[1+\frac{\{\ln(1-p)\}^2}{2}\mathbb{V}[v_p]\right]. \qquad (\text{A}\cdot 4)$$

Note that $\mathbb{E}[v_p] = |C|\times V_p$.

Equation (A·4) indicates that the probability of finding a target file, S_p, increases as the mean value of the number of the visited nodes, $\mathbb{E}[v_p]$, increases or the variance of the number of the visited nodes, $\mathbb{V}[v_p]$, decreases. Now, we can explain why both N_2 and N_3 find target files with almost the same probability though the mean value of the number of visited nodes in N_3 is larger than that in N_2. That is, we can find that the following two properties cancel each other.

- N_3 has a better property than N_2 to increase the probability of finding a target file because the mean value of the number of visited nodes in N_3 is larger than that in N_2.
- N_2 has a better property than N_3 to increase the probability of finding a target file because the variance of the number of visited nodes in N_2 is smaller than that in N_3.

So, we can resolve the seeming paradox pointed out in the last paragraph in Sect. 4.2.

Note that the value of $\{\ln(1-p)\}^2$ for $p = 0.01$ is smaller than that for $p = 0.05$. This means that the influence of $\mathbb{V}[v_p]$ in Eq. (A·4) is smaller when $p = 0.01$. So, we can say that the probability of finding a target file depends only on the mean value of the number of visited nodes when $p = 0.01$.