2016 IEEE 16th International Conference on Data Mining

Fractality of Massive Graphs: Scalable Analysis with Sketch-Based Box-Covering Algorithm*

Takuya Akiba†, Kenko Nakamura‡, Taro Takaguchi§
† *Preferred Networks, Inc.* ‡ *Recruit Communications Co., Ltd.*
§ *National Institute of Information and Communications Technology*
akiba@preferred.jp, kenkoooo@r.recruit.co.jp, taro.takaguchi@nict.go.jp

***Abstract*—Analysis and modeling of networked objects are fundamental pieces of modern data mining. Most real-world networks, from biological to social ones, are known to have common structural properties. These properties allow us to model the growth processes of networks and to develop useful algorithms. One remarkable example is the fractality of networks, which suggests the self-similar organization of global network structure. To determine the fractality of a network, we need to solve the so-called box-covering problem, where preceding algorithms are not feasible for large-scale networks. The lack of an efficient algorithm prevents us from investigating the fractal nature of large-scale networks. To overcome this issue, we propose a new box-covering algorithm based on recently emerging sketching techniques. We theoretically show that it works in near-linear time with a guarantee of solution accuracy. In experiments, we have confirmed that the algorithm enables us to study the fractality of million-scale networks for the first time. We have observed that its outputs are sufficiently accurate and that its time and space requirements are orders of magnitude smaller than those of previous algorithms.**

I. Introduction

Graph representation of real-world systems, such as social relationship, biological reactions, and hyperlink structure, gives us a strong tool to analyze and control these complex objects [19]. For the last two decades, we have witnessed the spark of network science that unveils common structural properties across a variety of real networks. We can exploit these frequently observed properties to model the generation processes of real networked systems [18] and to develop graph algorithms that are applicable to various objects [17]. A notable example of such properties is the scale-free property [3], [6], which manifests a power-law scaling in the vertex degree distribution and existence of well-connected vertices (often called hubs). The scale-free property, existence of hubs especially, underlies efficient performance of practical graph algorithms on realistic networks [1], [16].

Although the scale-free property inspires us to design better network models and algorithms, it is purely based on the local property of networks, i.e., the vertex degree. Real-world networks should possess other common properties beyond the local level. As a remarkable example of such non-local properties, the fractality of complex networks was found in network science [24], [13]. The fractality of a network suggests that the network shows a self-similar structure; if we replace groups of adjacent vertices with supervertices, the resultant network holds a similar structure to the original network (see Section II-B for its formal definition). The fractality of networks gives us unique insights into modeling of growth processes of real-world networks [25]. In addition, fractal and non-fractal networks, even with the same degree distribution, indicate striking differences in facility of spreading [22] and vulnerability against failure [15]. Aside from theoretical studies, the fractality provides us with useful information about network topology. Examples include the backbone structure of networks [14] and the hierarchical organization of functional modules in the Internet [7], metabolic [25] and brain [12] networks, to name a few.

Determination of the fractality of a network is based on the so-called box-covering problem [24] (also see Section II-B). We locally cover a group of adjacent vertices with a box such that all vertices in a box are within a given distance from each other, and then we count the number of boxes we use to cover the whole network. In principle, we have to minimize the number of boxes that cover the network, which is known to be an NP-hard problem (see [23] and references therein). Although different heuristic algorithms are proposed in the previous work (e.g., [23], [21]), they are still not so efficient as to be able to process networks with millions of vertices. This limitation leaves the fractal nature of large-scale networks far from our understanding.

Contributions: The main contribution of the present study is to propose a new type of box-covering algorithm that is much more scalable than previous algorithms. In general, previous algorithms first explicitly instantiate all boxes and then reduce the box cover problem to the famous set cover problem. This approach requires quadratic $\Theta(n^2)$ space for representing neighbor sets and is obviously infeasible for large-scale networks with millions of vertices. In contrast, the central idea underlying the proposed method is to solve the problem in the *sketch space*. That is, we do not explicitly instantiate neighbor sets; instead, we construct and use the *bottom-k min-hash sketch representation* [8], [9] of boxes.

Technically, we introduce several new concepts and algorithms. First, to make the sketch-based approach feasible, we introduce a slightly relaxed problem called $(1-\epsilon)$-BoxCover. We also define a key subproblem called the $(1-\epsilon)$-SetCover problem. The proposed box-cover al-

* Work done while all authors were at National Institute of Informatics.

2374-8486/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDM.2016.55

gorithm consists of two parts. First, we generate min-hash sketches of all boxes to reduce the $(1-\epsilon)$-BoxCover problem to the $(1-\epsilon)$-SetCover problem. Our sketch generation algorithm does not require explicit instantiation of actual boxes and is efficient in terms of both time and space. Second, we apply our efficient sketch-space set-cover algorithm to obtain the final result. Our sketch-space set-cover algorithm is based on a greedy approach, but is carefully designed with event-driven data structure operations to achieve near-linear time complexity.

We theoretically guarantee both the scalability and the solution quality of the proposed box-cover algorithm. Specifically, for a given trade-off parameter k and radius parameter ℓ, it works in $O((n+m)k \log k \min\{\ell, \log n\})$ time and $O(nk+m)$ space. The produced result is a solution of $(1-\epsilon)$-BoxCover within a factor $1+2\ln n$ of the optimum for BoxCover for $\epsilon \geq 2\sqrt{5(\ln n)/k}$, with a high probability that asymptotically approaches 1.

In experiments, we have confirmed the practicability of the proposed method. First, we observed that its outputs are quite close to those of previous algorithms and are sufficiently accurate to recognize networks with ground-truth fractality. Second, the time and space requirements are orders of magnitude smaller than those of previous algorithms, resulting in the capability of handling large-scale networks with tens of millions of vertices and edges. Finally, we applied our algorithm to a real-world million-scale network and accomplish its fractality analysis for the first time.

Repeatability: Our implementation of the proposed and previous box-cover algorithms is available at http://git.io/fractality. It also contains the generators of the synthetic network models, and thus the results in this paper can be perfectly replicated.

II. Preliminaries

We focus on networks that are modeled as undirected unweighted graphs. Let $G=(V,E)$ be a graph, where V and E are the vertex set and edge set, respectively. We use n and m to denote $|V|$ and $|E|$, respectively. For $d \geq 0$ and $v \in V$, we define $N_d(v)$ as the set of vertices with distance at most d from v. We call $N_d(v)$ the *d-neighbor*. When $d=1$, we sometimes omit the subscript, i.e., $N(v)=N_1(v)$. We also define $N_d(S)$ for a set $S \subseteq V$ as $N_d(S)=\bigcup_{v \in S} N_d(v)$. In other words, $N_d(S)$ represents the set of vertices with distance at most d from at least one vertex in S.

A. Bottom-k Min-Hash Sketch

In this subsection, we review the bottom-k min-hash sketch and its cardinality estimator [8], [9]. Let X denote the ground set of items. We first assign a random rank value $r_i \sim U(0,1)$ to each item $i \in X$, where $U(0,1)$ is the uniform distribution on $(0,1)$. Let S be a subset of X. For an integer $k \geq 1$, the bottom-k min-hash sketch of S is defined as $\widetilde{S}$, where $i \in \widetilde{S} \iff r_i \leq k\text{-th}\{r_j \mid j \in S\}$. In other words, $\widetilde{S}$ is the set of vertices with the k smallest rank values. We define $\widetilde{S}=S$ if $|S|<k$.

For a set $S \subseteq X$, the *threshold rank* $\tau(S)$ is defined as follows. If $|S| \geq k$, $\tau(S)=k\text{-th}\{r_i \mid i \in S\}$. Otherwise, $\tau(S)=(k-1)/|S|$. Note that $\tau(S)=\tau(\widetilde{S})$. Using sketch $\widetilde{S}$, we estimate the cardinality $|S|$ as $\widetilde{C}(S)=(k-1)/\tau(\widetilde{S})$. The cardinality estimation $\widetilde{C}(S)$ is an unbiased estimator of $|S|$, and has a coefficient of variation (CV)[1] of at most $1/\sqrt{k-2}$ [8], [9]. Moreover, for $\epsilon>0$ and $c>1$, by setting $k \geq (2+c)\epsilon^{-2}\ln|X|$, the probability of the estimation having a relative error larger than ϵ is at most $1/|X|^c$ [10].

B. Problem Definition

Graph Fractality: The fractality of a network [24] is a generalization of the fractality of a geometric object in Euclidean space [11]. A standard way to determine the fractality of a geometric object is to use the so-called box-counting method; we tile the object with cubes of a fixed length and count the number of cubes needed. If the number of cubes follows a power-law function of the cube length, the object is said to be fractal. A fractal object holds a self-similar property so that we observe similar structure in it when we zoom in and out to it.

The idea of the box counting method is generalized to analyze the fractality of networks [24]. The box-covering method for a network works by covering the network with boxes of finite length ℓ, which refers to a subset of vertices in which all vertices are within distance ℓ. For example, a box with $\ell=1$ is a set of nodes all adjacent to each other. If the number of boxes of length ℓ needed to cover the whole network, denoted by $b(\ell)$, follows a power-law function of ℓ: $b(\ell) \propto \ell^{-d}$, the network is said to be fractal. The exponent d is called the fractal dimension. As can be noticed, $b(\ell)$ crucially depend on how we put the boxes. Theoretically, we have to put boxes such that $b(\ell)$ is minimized to assess its precise scaling. However, this box-covering problem is NP-hard and that is why we propose our new approximation algorithm of this problem in the rest of this paper.

Box Cover: As we described in the previous section, the fractality of graphs is analyzed by solving the box covering problem. The problem has two slightly different versions: the diameter version [24] and the radius version [23]. It has been empirically shown that these two versions yield negligible difference in the results. In this study, we focus on the radius version: In the BoxCover problem, given a graph G and a radius limit $\ell>0$, the objective is to find a set $S \subseteq V$ of the minimum size such that $N_\ell(S)=V$.

The size of set S is equal to $b(\ell)$ discussed in the last section. In this study, we consider a slightly relaxed variant of the BoxCover problem, named $(1-\epsilon)$-BoxCover. In the $(1-\epsilon)$-BoxCover problem, we are given a graph G, a radius limit $\ell>0$ and an error tolerance parameter $\epsilon>0$. The objective is to find a set $S \subseteq V$ of the minimum size such that $|N_\ell(S)| \geq (1-\epsilon)n$.

Set Cover: The BoxCover problem is a special case of the SetCover problem. In the SetCover problem, we are given

[1]The CV is the ratio of the standard deviation to the mean.

a set family $\{S_p\}_{p \in P}$. The objective is to find a set $R \subseteq P$ of the minimum size such that $\bigcup_{p \in R} S_p = \bigcup_{p \in P} S_p$.

The proposed box-covering algorithm deals with a slightly different version of SETCOVER, named $(1-\epsilon)$-SETCOVER *with sketched input* as a key subproblem. In the sketched input version of the $(1-\epsilon)$-SETCOVER problem, we are given the min-hash sketches of a set family $\{\widetilde{S}_p\}_{p \in P}$ and an error tolerance parameter $\epsilon > 0$. The objective is to find a set $R \subseteq P$ of the minimum size such that $|\bigcup_{p \in R} S_p| \geq (1-\epsilon)|\bigcup_{p \in P} S_p|$.

We first design an efficient approximation algorithm for $(1-\epsilon)$-SETCOVER (Section III). We then propose a new box-covering algorithm using it (Section IV).

III. SET COVER IN SKETCH SPACE

In this section, we design an efficient approximation algorithm for the sketched-input version of $(1-\epsilon)$-SETCOVER. We call each $p \in P$ a *collection* and $i \in S_p$ an *element*. Because of the connection to the BOXCOVER problems, we assume that the numbers of collections and elements are equal. We denote them by n, that is, $|P| = |\bigcup_{p \in P} S_p| = n$. For $R \subseteq P$, we define $S_R = \bigcup_{p \in R} S_p$. Moreover, for simplicity, we denote $\widetilde{C}(S_R)$ by $\widetilde{C}(R)$, which can be calculated from merged min-hash sketch $\widetilde{S}_R$.

We first explain the basic greedy algorithm that runs in $O(n^2 k)$ time, and then present its theoretical solution guarantee. Finally, we propose an efficient greedy algorithm, which runs in $O(nk \log n)$ time and produces the exact same solution as the basic algorithm.

A. Basic Greedy Algorithm

Our basic greedy algorithm Select-Greedily-Naive works as follows. We start with an empty set $R = \{\}$. In each iteration, we calculate $\widetilde{C}(R \cup \{p\})$ for every $p \in P \setminus R$, and select p that maximizes the estimated cardinality, and add it to R. We repeat this until $\widetilde{C}(R)$ gets at least $(1-\epsilon/2)n$, and the resulting R is the solution.

To calculate $\widetilde{C}(R \cup \{p\})$, together with R, we manage the merged min-hash sketch $\widetilde{S}_R$, so that $\widetilde{S}_R$ always corresponds to the min-hash sketch of S_R. To this end, we use the merger operation of min-hash sketch. Let us assume that the items in a min-hash sketch are stored in the ascending order of their ranks. Then, merging two min-hash sketches can be done in $O(k)$ time like in the merge sort algorithm; we just need to pick the top-k distinct items with the lowest ranks in the two min-hash sketches. Algorithm Select-Greedily-Naive runs in $O(n^2 k)$ time and $O(nk)$ space.

B. Theoretical Solution Guarantee

We can guarantee the quality of the solution produced by the above algorithm as follows.

Lemma 1: *For $\epsilon \geq 2\sqrt{5(\ln n)/k}$, algorithm Select-Greedily-Naive produces a solution of $(1-\epsilon)$-SETCOVER within a factor $1 + 2\ln n$ of the optimum for SETCOVER with a probability of at least $1 - 1/n$.*

In other words, with a high probability that asymptotically approaches 1, R is the solution of $(1-\epsilon)$-SETCOVER and $|R| \leq (1 + 2\ln n)|R^*|$, where R is the output of algorithm Select-Greedily-Naive and R^* is the optimum solution of SETCOVER (with the same set family as the input). The proof is in our full version [2].

C. Near-Linear Time Greedy Algorithm

Algorithm Select-Greedily-Naive takes quadratic time, which is unacceptable for large-scale set families. Therefore, we then design an efficient greedy algorithm Select-Greedily-Fast, which produces the exact same output as algorithm Select-Greedily-Naive but runs in $O(nk \log n)$ time and $O(nk)$ space. As the input size is $O(nk)$, this algorithm is near-linear time.

The behavior of Select-Greedily-Fast at a high level is the same as that of Select-Greedily-Naive. That is, we start with an empty set $R = \{\}$, and, at each iteration, it adds $p \in P \setminus R$ with the maximum gain on $\widetilde{C}$ to R. The central idea underlying the speed-up is to classify the state of each $p \in P$ at each iteration into two types and manage differently to reduce the reevaluation of the gain. Due to the space constraint, please refer to our full version [2] for details.

IV. SKETCH-BASED BOX COVERING

In this section, we complete our sketch-based box-covering algorithm for the $(1-\epsilon)$-BOXCOVER problem. We first propose an efficient algorithm to construct min-hash sketches representing the ℓ-neighbors, and then present and analyze the overall box-covering algorithm,

A. Sketch Generation

For $v \in V$, we denote the min-hash sketch of $N_\ell(v)$ as $\widetilde{N}_\ell(v)$. Here, we construct $\widetilde{N}_\ell(v)$ for all vertices $v \in V$ to reduce the $(1-\epsilon)$-BOXCOVER problem to the $(1-\epsilon)$-SETCOVER problem. Our sketch construction algorithm Build-Sketches is as follows (please refer to [2] for details).

It receives a graph G and a radius parameter ℓ. Each vertex v manages a tentative min-hash sketch X_v. Initially, X_v only includes the vertex itself, i.e., $X_v = \{v\}$, which corresponds to $\widetilde{N}_0(v)$. Then, we repeat the following procedure for ℓ times so that, after the i-th iteration, $X_v = \widetilde{N}_i(v)$.

In each iteration, for each vertex, we essentially merge the sketches of its neighbors into its sketch in a message-passing-like manner. Two speed-up techniques are employed here to avoid an unnecessary insertion check. For $v \in V$, let A_v be the vertices in whose sketches v is added to in the last iteration. First, for each $v \in V$, we try to insert v only into the sketches of the vertices that are neighbors of A_v, as v cannot be inserted into other vertices. Second, we conduct the procedure above in the increasing order of ranks, since this decreases the unnecessary insertion. Algorithm Build-Sketches runs in $O((n+m)k \log k \min\{\ell, \log n\})$ expected time and $O(nk+m)$ space. The proof is in [2].

B. Overall Box-Cover Algorithm

The overall box-covering algorithm Sketch-Box-Cover is as follows. We first construct the min-hash sketches using algorithm Build-Sketches and then solve the set cover problem in the sketch space using algorithm Select-Greedily-Fast.

Theorem 2 (Scalability guarantee)**:** *Algorithm Sketch-Box-Cover works in $O((n+m)k \log k \min\{\ell, \log n\})$ time and $O(nk+m)$ space.*

Theorem 3 (Solution accuracy guarantee)**:** *With a probability of at least $1-1/n$, for $\epsilon \geq 2\sqrt{5(\ln n)/k}$, algorithm Sketch-Box-Cover produces a solution to the $(1-\epsilon)$-BOXCOVER problem within a factor $1+2\ln n$ of the optimum for the BOXCOVER problem.*

Assuming k is a constant, the time and space complexities are near-linear. Similarly, given a constant ϵ, the time and space complexities are still near-linear, since it suffices to set $k = \lceil 20\epsilon^{-2} \ln n \rceil$. In practice, as seen in our experiments, the algorithm produces solutions that are much closer to the optimum than what is expected from the above approximation ratio with much smaller k.

V. PRACTICAL IMPROVEMENT

In this section, we propose techniques to improve the practicality of the proposed method.

Exact Coverage Management: For the termination condition in the greedy selection algorithm, we propose to use the exact coverage $C(R)$ instead of the estimated coverage $\widetilde{C}(R)$. This technique makes the results more stable. With some techniques, the total time consumption of coverage management is $O((n+m)\ell)$ (see [2] for details).

Multi-Pass Execution: On the basis of the above exact coverage management technique, we sometimes detect that, even while the estimated coverage is saturated (i.e., $\widetilde{C}(R) = \widetilde{C}(P)$), the actual coverage is below the specified threshold. In that case, to choose more vertices, we propose to repeat the algorithm from sketch construction until the actual coverage becomes higher than the threshold.

Exact Neighborhood: To further improve the accuracy, we propose to combine our sketch-based algorithm with a non-sketch-based algorithm. For a very small radius parameter ℓ, neighborhood $N_\ell(v)$ is sometimes much smaller than k. Moreover, even for a larger ℓ, when using the above multi-pass execution technique, the remaining neighbors may become small in later passes. In these cases, the sketching approach has little advantage. Therefore, we detect such circumstances and switch to a non-sketch-based greedy algorithm.

Exact Box Covering: Together with the preceding three techniques, to further make the results reliable, we propose to use our algorithm for solving the original BOXCOVER problem rather than $(1-\epsilon)$-BOXCOVER problem. In other words, we recommend setting $\epsilon = 0$ to ensure that all vertices are completely covered. As we will see in the experiments, even with this seemingly extreme threshold, thanks to the above techniques, both the running time and the solution quality are reasonable.

VI. EXPERIMENTS

In this section, we present our experimental results to verify the performance of the algorithm. Specifically, we compare our algorithm with other preceding algorithms in terms of accuracy and computation time. For more results, please see [2].

A. Setup

Experiments were conducted on a Linux server with Intel Xeon X5650 (2.67 GHz) and 96GB of main memory. Algorithms were implemented in C++ and compiled by gcc 4.8.4 with `-O3` option. For comparison, we used a naive algorithms named greedy coloring (GC) and three advanced and popular algorithms, named maximum excluded mass burning (MEMB), minimal value burning (MVB), and compact box burning (CBB). GC, MEMB, and CBB were introduced in [23] and MVB was in [21].

Network Models: We used two network models with ground-truth fractality: the (u,v)-flower [20] and the Song-Havlin-Makse (SHM) [25] model. Both models can be either fractal or non-fractal, depending on the structural parameter values. We refer to them as the (u,v,g)-flower and (c,e,g)-SHM model to indicate the parameter settings. The common parameter g ($g = 1,2,3,\dots$) determines the network size n: $n = (w-2/w-1)w^g + w/w-1$, where $w \equiv u+v$ for the (u,v,g)-flower, and $n = (2c+1)^g n_0$ for the (c,e,g)-SHM model. In addition to the flower and SHM models, we considered the Barabási-Albert (BA) network model [3] as one of the most famous models of complex networks. The BA model is not fractal [24]. We refer to this model as (c,t)-BA, where c is the number of edges that a new node has and t sets the network size as $n = 125 \times 2^t$.

Fractality Decision Procedure: After the computation of the box-covering algorithms, we determined whether the obtained $b(\ell)$ indicates the fractality or not. This task was done by fitting the $b(\ell)$ curve with a power-law function (i.e., fractal) and an exponential function (i.e., non-fractal) by using `optimize.leastsq` function in `SciPy` package of Python. The key quantity was the ratio between the residual error of fitting to a power-law function and that to an exponential function, denoted by r_{fit}. If $-\log_{10} r_{\mathrm{fit}}$ is postive (i.e., $r_{\mathrm{fit}} < 1$), the network was supposed to be fractal. Otherwise, it was supposed to be non-fractal. This procedure of fitting and comparison follows that used in [26].

B. Parameter Settings

First of all, we have to decide the parameter values of our algorithm: ϵ (error tolerance), k (sketch size), and α (exact neighborhood switch threshold). In principle, the accuracy of results as well as running time increases with k and α, and it decreases with ϵ. As we discussed in Section V, we fixed $\epsilon = 0$. To choose k and α, we plotted the average

TABLE I

RUNNING TIME IN SECONDS (*Time*) AND THE RELATIVE ERROR RATIO OF A POWER-LAW FUNCTION, $-\log_{10} r_{\text{fit}}$ (*Fit*). DNF MEANS THAT IT DID NOT FINISH IN ONE DAY OR RAN OUT OF MEMORY.

Model	Graph $\lvert V\rvert$	$\lvert E\rvert$	Sketch Time	Sketch Fit	MEMB [23] Time	MEMB [23] Fit	GC [23] Time	GC [23] Fit	MVB [21] Time	MVB [21] Fit	CBB [23] Time	CBB [23] Fit
▽ ***Networks with ground-truth fractality*** ("Fit" values are expected to be positive.)												
(2, 2, 4)-flower	172	256	0	0.8	0	1.0	0	28.7	199	1.0	0	28.0
(2, 2, 7)-flower	10,924	16,384	15	2.5	10	3.4	228	27.7	DNF	—	122	27.4
(2, 2, 11)-flower	2,796,204	4,194,304	62,138	4.0	DNF	—	DNF	—	DNF	—	DNF	—
(2, 3, 6)-flower	11,720	15,625	26	1.2	14	1.1	146	0.1	DNF	—	5,593	0.5
(2, 3, 8)-flower	292,970	390,625	2,913	1.0	DNF	—	DNF	—	DNF	—	DNF	—
(3, 3, 6)-flower	37,326	46,656	148	1.1	101	1.3	10,751	2.1	DNF	—	1,284	1.4
(3, 3, 7)-flower	223,950	279,936	1,779	1.2	DNF	—	DNF	—	DNF	—	61,562	1.5
(3, 4, 5)-flower	14,007	16,807	34	0.7	16	0.9	560	0.1	DNF	—	3,380	-0.4
(3, 4, 7)-flower	686,287	823,543	8,873	0.8	DNF	—	DNF	—	DNF	—	DNF	—
(2, 0, 6)-SHM	12,501	12,500	33	1.2	8	1.1	872	1.1	32	1.1	325	0.7
(2, 0, 8)-SHM	312,501	312,500	2,728	1.1	DNF	—	DNF	—	DNF	—	DNF	—
▽ ***Networks with ground-truth non-fractality*** ("Fit" values are expected to be negative.)												
(1, 2, 10)-flower	29,526	59,049	108	-2.9	197	-2.9	286	-5.4	364	-2.2	21,833	-2.6
(1, 2, 12)-flower	265,722	531,441	1,774	-4.6	DNF	—	38,278	-7.0	DNF	—	DNF	—
(1, 3, 7)-flower	10,924	16,384	20	-3.0	16	-2.7	44	-4.8	DNF	—	1,862	-3.3
(1, 3, 9)-flower	699,052	1,048,576	4,195	-6.0	DNF	—	DNF	—	DNF	—	DNF	—
(1, 4, 6)-flower	11,720	15,625	23	-1.0	20	-0.6	53	-1.8	DNF	—	3,781	-1.9
(1, 4, 8)-flower	292,970	390,625	1,678	-0.7	DNF	—	67,866	-1.9	DNF	—	DNF	—
(2, 1, 6)-SHM	24,885	31,104	31	-3.5	32	-3.5	433	-0.4	126	-3.5	7,129	-2.5
(2, 1, 7)-SHM	149,301	186,624	390	-4.9	1,397	-4.9	17,703	-0.4	8,615	-4.9	DNF	—
(3, 1, 5)-SHM	14,045	16,384	12	-2.6	8	-2.6	97	-0.3	25	-2.6	1,224	-2.9
(3, 1, 6)-SHM	112,349	131,072	210	-4.1	580	-4.2	9,504	-0.3	2,070	-4.2	DNF	—
(2, 1)-BA	250	497	0	-0.9	0	-0.9	0	-0.6	54	-0.5	0	-0.3
(2, 4)-BA	2,000	3,997	1	-2.7	0	-2.0	2	-0.6	DNF	—	404	-0.1
(2, 10)-BA	128,000	255,997	377	-1.5	3,535	-1.5	12,457	-0.6	DNF	—	DNF	—
(2, 13)-BA	1,024,000	2,047,997	6,474	-1.4	DNF	—	DNF	—	DNF	—	DNF	—
(2, 15)-BA	4,096,000	8,191,997	36,125	-1.4	DNF	—	DNF	—	DNF	—	DNF	—

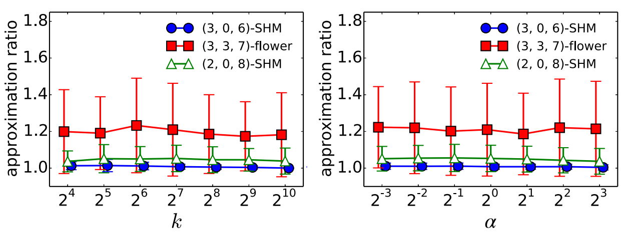

Fig. 1. Average approximation ratio to the theoretical solutions for various $\boldsymbol{k}$ and $\boldsymbol{\alpha}$.

approximation ratio of our results to the theoretical solutions for several fractal network models as a function of k and α in Figure 1. The average approximation ratio is defined by $\rho \equiv \langle b_{\text{sketch}}(\ell)/b_{\text{theory}}(\ell)\rangle_\ell$, where $\langle\cdot\rangle_\ell$ is the average over ℓ. To compute $b_{\text{sketch}}(\ell)$, we executed the algorithm for ten times and took the average of the resulting $b(\ell)$ over the ten runs.

In the left panel of Figure 1, we varied $2^4 \leq k \leq 2^{10}$ while fixing $\alpha = 1$. The ρ values were affected slightly by k for the SHM models and tended to decrease with k for the flower network. On the basis of the results, we decided to use $k = 2^7$ throughout the following experiments. In the right panel of Figure 1, we varied $2^{-3} \leq \alpha \leq 2^3$ while fixing $k = 2^7$. The ρ values were almost constant regardless of the α values for all of the three networks considered. Therefore, taking into account the running time, we decided to use $\alpha = 1$ throughout the following experiments.

C. Accuracy and Scalability

Table I summarizes the main results of this paper and shows the comparison of our algorithm (*Sketch*) with other preceding algorithms for fractal and non-fractal network models with various sizes. We evaluated the performance of algorithms by two measures. The first was the accuracy given by $-\log_{10} r_{\text{fit}}$ (Section VI-A). If this measure took a positive (negative) value for a fractal (non-fractal) network, the algorithm correctly distinguished the fractality of the network. The second was computation time in seconds.

Discrimination Ability: As we can see in Table I, the sketch algorithm perfectly distinguishes between the fractal and non-fractal networks as the other algorithms do (except for CBB for $(3, 4, 5)$-flower). The proposed algorithm shows its advantage in computation time: the algorithm is generally faster than other algorithms and is able to handle large networks that other algorithms do not terminate. Although MEMB is faster than Sketch for some relatively small network models, this result is expected because actual neighborhood sets are not significantly larger than sketch sizes in these networks. As a summary, *(i)* the sketch algorithm correctly detected the fractality of network models with around ten times smaller computation time than the fastest previous algorithm. In addition, *(ii)* the algorithm was able to deal with networks with millions of nodes with acceptable computation time (within 1 day), whereas other algorithms could not in our machine environment.

Time and Memory Consumption: The proposed algorithm is scalable for not only for computation time but also for memory usage. In Figure 2, computation time (seconds) and memory usage (KB) of the five algorithms were plotted as a function of the number of vertices. We use $(2, 2, g)$-flower $(3 \leq g \leq 11)$ and $(2, t)$-BA $(0 \leq t \leq 15)$ networks as the example of a fractal and a non-fractal network, respectively. The symbols corresponding to an algorithm were not shown if the algorithm did not stop within 24 hours or could not

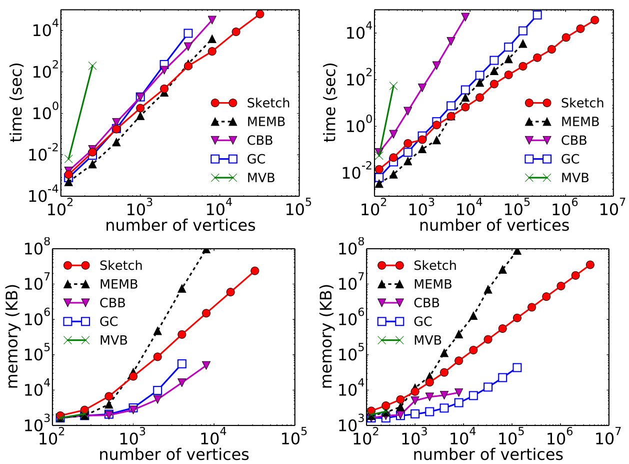

Fig. 2. Scalability of computation time (top) and memory usage (bottom) for $(2, 2, g)$-flower (left) and $(2, t)$-BA (right) networks.

execute owing to memory shortage. The performance of the proposed algorithm is comparable to or worse than some other algorithms when the network is relatively small (i.e., $n < 10^4$). However, the algorithm is orders of magnitude faster than other algorithms for large networks. Also, it achieves such high a high speed with incomparably smaller memory usage than MEMB, the second fastest algorithm.

D. Application to Real Large Network

In closing this section, we applied the sketch algorithm to a large-scale real graph to show the scalability of the proposed algorithm with an empirical instance. The results also gave us some insight on the fractality of large-scale real-world networks, which is beyond the reach of previous algorithms. As a representative instance of a real-world large graph, we considered the `in-2004` network [5], [4], which is a crawled web graph of $1,382,908$ vertices and $16,917,053$ edges. We discarded the direction of the edges (i.e., hyperlinks) to make the network undirected. The algorithm took 11.7 hours in total.

The resulting $b(\ell)$ of the sketch algorithm and the fitting curves are shown in Figure 3. We omitted the three points with the smallest ℓ values from the fitting because empirical networks would not show a perfect fractality, contrary to well-designed network models. A large part of the points fall on the line of the fitted power-law function, and indeed, our fractality decision procedure yielded $-\log_{10} r_{\text{fit}} = 0.79$, which suggests the fractality of the `in-2004` network. It is worth mentioning that the fractality of this network was unveiled for the first time for the sake of our algorithm.

Acknowledgment: This work was supported by JSPS KAKENHI (No. 15H06828), JST, ERATO, Kawarabayashi Large Graph Project, and JST, PRESTO. Web graph data was downloaded from http://law.di.unimi.it/datasets.php. T.T. thanks to K. Takemoto for valuable discussions.

References

[1] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, page 349, 2013.

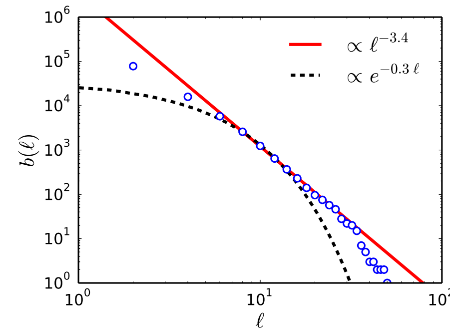

Fig. 3. Results for a real web graph.

[2] T. Akiba, K. Nakamura, and T. Takaguchi. Fractality of massive graphs: Scalable analysis with sketch-based box-covering algorithm. *CoRR*, abs/1609.07994, 2016.
[3] A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.
[4] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596, 2011.
[5] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, pages 595–601, 2004.
[6] G. Caldarelli. *Scale-Free Networks*. Oxford University Press, 2007.
[7] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of Internet topology using k-shell decomposition. *Proc. Natl. Acad. Sci. USA*, 104(27):11150–11154, 2007.
[8] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.
[9] E. Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. *IEEE TKDE*, 27(9):2320–2334, 2015.
[10] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM*, pages 629–638, 2014.
[11] K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications, Second Edition*. Wiley-Blackwell, 2003.
[12] L. K. Gallos, H. A. Makse, and M. Sigman. A small world of weak ties provides optimal global integration of self-similar modules in functional brain networks. *Proc. Natl. Acad. Sci. USA*, 109(8):2825–2830, 2012.
[13] L. K. Gallos, C. Song, and H. A. Makse. A review of fractality and self-similarity in complex networks. *Physica A*, 386:686–691, 2007.
[14] K.-I. Goh, G. Salvi, B. Kahng, and D. Kim. Skeleton and Fractal Scaling in Complex Networks. *Phys. Rev. Lett.*, 96(1):018701, 2006.
[15] T. Hasegawa and K. Nemoto. Hierarchical scale-free network is fragile against random failure. *Phys. Rev. E*, 88(6):062807, 2013.
[16] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.
[17] J. Kleinberg. The small-world phenomenon. In *STOC*, pages 163–170, 2000.
[18] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over Time : Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*, pages 177–187, 2005.
[19] M. E. J. Newman. *Networks: an Introduction*. Oxford University Press, 2010.
[20] H. D. Rozenfeld, S. Havlin, and D. Ben-Avraham. Fractal and transfractal recursive scale-free nets. *New J. Phys.*, 9(6):175, 2007.
[21] C. M. Schneider, T. A. Kesselring, J. S. Andrade, and H. J. Herrmann. Box-covering algorithm for fractal dimension of complex networks. *Phys. Rev. E*, 86(1):016707, 2012.
[22] M. Á. Serrano, D. Krioukov, and M. Boguñá. Percolation in Self-Similar Networks. *Phys. Rev. Lett.*, 106(4):048701, 2011.
[23] C. Song, L. K. Gallos, S. Havlin, and H. A. Makse. How to calculate the fractal dimension of a complex network: the box covering algorithm. *J. Stat. Mech.*, 2007(03):P03006, 2007.
[24] C. Song, S. Havlin, and H. A. Makse. Self-similarity of complex networks. *Nature*, 433(7024):392–395, 2005.
[25] C. Song, S. Havlin, and H. A. Makse. Origins of fractality in the growth of complex networks. *Nat. Phys.*, 2(4):275–281, 2006.
[26] K. Takemoto. Metabolic networks are almost nonfractal: A comprehensive evaluation. *Phys. Rev. E*, 90(2):022802, 2014.