INVITED PAPER

Sublinear Computation Paradigm: Constant-Time Algorithms and Sublinear Progressive Algorithms

Kyohei CHIBA[†], ***Nonmember and*** **Hiro ITO**[†a)], ***Member***

SUMMARY The challenges posed by big data in the 21st Century are complex: Under the previous common sense, we considered that polynomial-time algorithms are practical; however, when we handle big data, even a linear-time algorithm may be too slow. Thus, sublinear- and constant-time algorithms are required. The academic research project, "Foundations of Innovative Algorithms for Big Data," which was started in 2014 and will finish in September 2021, aimed at developing various techniques and frameworks to design algorithms for big data. In this project, we introduce a "Sublinear Computation Paradigm." Toward this purpose, we first provide a survey of constant-time algorithms, which are the most investigated framework of this area, and then present our recent results on sublinear progressive algorithms. A sublinear progressive algorithm first outputs a temporary approximate solution in constant time, and then suggests better solutions gradually in sublinear-time, finally finds the exact solution. We present Sublinear Progressive Algorithm Theory (SPA Theory, for short), which enables to make a sublinear progressive algorithm for any property if it has a constant-time algorithm and an exact algorithm (an exponential-time one is allowed) without losing any computation time in the big-O sense.

key words: *sublinear computation paradigm, big data, sublinear-time algorithms, constant-time algorithms, tester, progressive algorithms*

1. Introduction

Big data is changing the way people live their everyday lives in all aspects of modern life from health and medicine to security and entertainment. Requirements for computer algorithms are also evolving, particularly in the need for speed. For example, in the past, polynomial-time algorithms were considered fast, but if we applied an $O(n^2)$-time algorithm on big data of a peta-byte scale or more, we would have encountered problems with computational resources or the running time. When we handle big data, even a linear-time algorithm may be too slow! Certainly, in the era of big data, we need sublinear-time algorithms. If the computation (= running) time of an algorithm is $o(n)$, where n is the size of the input, then the algorithm is called a *sublinear-time algorithm*. If the running time is constant (i.e., $O(1)$), then it is called a *constant-time algorithm*, which is a special case of a sublinear-time algorithm.

From this point of view, an academic research project, "Foundations of Innovative Algorithms for Big Data," whose research director is Naoki Katoh, Professor of Hyogo University, began in October, 2014 and will finish in September 2021*. The total budget for this project is more than 3 million dollars. Although the project was started by 24 members, many more researchers have gathered and now the number of regular members has grown to more than 40 in total.

In this project, we introduce a new paradigm,

"Sublinear Computation Paradigm,"

which shows how we need sublinear-time algorithms in various situations in the era of big data. Under this paradigm, we have obtained many fruitful results [26], [27], especially in the area of constant-time algorithms. The paper [20] presents a constant-time "universal" tester for a model of complex networks, and was selected as one of the best three works in the final report of the project.

In this paper, we survey sublinear-time, mainly constant-time, algorithms. In this area, property testing is the most examined framework. Property testing probabilistically distinguishes that the input has a predetermined property and that the input is far from satisfying the property. Property testing was firstly presented by Rubinfeld and Sudan [34] in 1992 in the context of program checking. The first study that presented the notion of the constant-time testability of combinatorial structures (mainly graphs) was given by Goldreich, Goldwasser, and Ron [17], whose conference version appeared in 1995 (STOC'95). Many studies following their idea of testability have appeared and the importance of this area is growing. See [8], [15], [16] for details.

This paper also introduces the idea of "sublinear progressive algorithms" and presents some results on how to construct these algorithms. A sublinear progressive algorithm first outputs a temporary approximate solution in constant time, and then suggests better solutions gradually, finally finding the exact solution. We present Sublinear Progressive Algorithm Theory (SPA Theory, for short; Theorem 23), which enables to make a sublinear progressive algorithm for any property if it has a constant-time algorithm and an exact algorithm (an exponential-time one is allowed) without losing any computation time in the big-O sense.

This paper is organized as follows. After presenting the basic notations and terminology in Sect. 2, we show some tools useful in this area in Sect. 3. Basic techniques are explained by using examples based on fundamental problems

Manuscript received March 27, 2021.
Manuscript revised August 30, 2021.
Manuscript publicized October 8, 2021.

[†]The authors are with School of Informatics and Engineering, The University of Electro-Communications, Chofu-shi, 182-8585 Japan.
a) E-mail: itohiro@uec.ac.jp

DOI: 10.1587/transfun.2021EAI0003

*The main project finished successfully in March 2020. Following this success, an additional project, whose scale has been reduced, will continue until September 2021.

in Sect 4. Next, we present key results in this area, focusing mainly on the characterizations of constant-time testable properties in Sect. 5. Finally, we show our results on sublinear progressive algorithms through detailed discussions in Sect. 6 and we present our conclusions in Sect. 7.

2. Notation and Terminology

2.1 Basic Terms and Symbols

In this paper, a graph is a simple graph, i.e., it has neither a self-loop nor parallel edges, unless otherwise stated. For simplicity, we omit rounding operators necessary to ensure that all values of formulas such as $\sqrt{n}$ are integers.

Let $\mathbb{Z}$ and $\mathbb{R}$ be the set of integers and real numbers, respectively. For any set of real numbers R, $R^+ := \{x \in R \mid x > 0\}$ and $R_0^+ := \{x \in R \mid x \geq 0\}$, e.g., $\mathbb{Z}_0^+$ is the set of nonnegative integers.

2.2 Oracles

A sublinear computation time means that an algorithm does not read the whole data of an input except for the case when that the size of the input is very small (i.e., smaller than some constant). Thus in order to consider sublinear-time algorithms, how to model the problems is important. A subliner-time algorithm gets the data of the input through an oracle. If an algorithm queries a question, then the oracle gives a constant-sized answer. For example, in the dense-graph model, which is one of the most-studied models, the *edge-oracle* is used: If an algorithm asks a pair of vertex ID's, say (i, j), then the oracle answers 1 if there is an edge between them; otherwise 0. Here, we normally assume that the ID of vertices are given by a set of successive positive integers from 1 to n, where n is the number of vertices and the algorithm knows n. Through this oracle, algorithms get (partial) information of the input graph.

Since an algorithm reads only a part of the input, getting a correct result is usually impossible. Thus we should introduce a relaxation. We explain about "property testing," which is the most studied framework in subliner-time algorithms.

In this framework, we allow an approximation error: An algorithm for testing a property accepts the input with high probability (say more than $2/3$) if it has the property, or rejects the input with high probability if it is far from having the property. To treat this idea mathematically, we must define what a property is and what "far" means.

2.3 Distance and ϵ-Far

The above "far" is quantified by using a positive real number $\epsilon > 0$: To explain the farness, we use graphs for example. For other types of inputs, e.g., functions, grammars, strings, images, and figures, similar methods as graphs are used.

We introduce the distance between two instances (= inputs). We can define the distance only between two instances whose size of the input, say N, are the same. Note that since an input is represented by the answers of an oracle, N is equal to the number of possible different queries, e.g., for the edge-oracle, $N = n^2$, where n is the number of vertices[†]. In other words, if the inputs are graphs, then we define the distance only between graphs whose number of vertices are the same. We call a graph that consists of n vertices an *n-graph*.

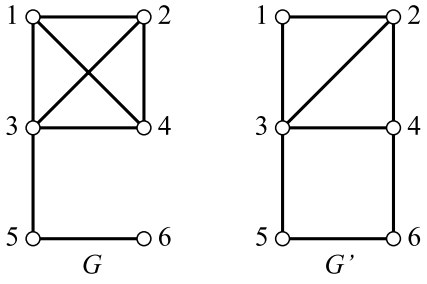

Fig. 1 $\ell(G, G') = 2$.

The distance between two instances I and I' the size of which are both N, denoted by $\mathrm{dist}(I, I')$, is defined as follows. Let $\ell(I, I')$ be the number of distinct queries of the oracle whose answers are different between I and I', e.g., if $I = G = (V, E)$ and $I' = G' = (V, E')$ are n-graphs and the oracle is the edge-oracle, then $\ell(G, G')$ is the number of pairs $(i, j) \in n \times n$ such that $(i, j) \in E \wedge (i, j) \notin E'$ or $(i, j) \notin E \wedge (i, j) \in E'$. See Fig. 1 for an example: $\ell(G, G') = 2$ since removing $(1, 4)$ and adding $(4, 6)$ are necessary to make G equal to G'.

Now,

$$\mathrm{dist}(I, I') := \frac{\ell(I, I')}{N}. \tag{1}$$

Next, we define the distance between an instance and a property. Before showing the definition, we define properties. Let Π be the universal set of possible instances. Let Π_N be the subset of instances whose size is N in Π. Clearly $\Pi = \bigcup_{i=1}^{\infty} \Pi_i$. A *property* is a subset of Π closed under isomorphism. The intuitive meaning of how those two instances are isomorphic is that they are the same except for their labels (IDs), e.g., when we consider graphs, two n-graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijection $\pi : V \to V'$ such that $(i, j) \in E \Leftrightarrow (\pi(i), \pi(j)) \in E'$. For an example of properties, a graph property "planar" is defined by the set of planar graphs. Note that this is clearly closed under isomorphism. For any property $\mathcal{P}$, $\mathcal{P}_i := \mathcal{P} \cap \Pi_i$.

The distance between an instance I and a property $\mathcal{P}$ is defined as follows. Let N be the size of I.

$$\mathrm{dist}(I, \mathcal{P}) := \begin{cases} \min_{I' \in \mathcal{P}_N} \mathrm{dist}(I, I') & \text{if } \mathcal{P}_N \neq \emptyset, \\ \infty & \text{otherwise.} \end{cases}$$

Let I and I' be instances and let $\mathcal{P}$ be a property. For a

[†]If the input is an undirected simple graph, then it must be $\binom{n}{2} = n(n-1)/2$. We normally use, however, n^2 for simplicity. Note that multiplying any constant has essentially no effect.

positive real number $\epsilon > 0$, we say that I and I' are *ϵ-far* if $\mathrm{dist}(I, I') > \epsilon$; otherwise, *$\epsilon$-close*. We also say that I and $\mathcal{P}$ are *ϵ-far* if $\mathrm{dist}(I, \mathcal{P}) > \epsilon$; otherwise, *$\epsilon$-close*.

2.4 Testers

We define testing algorithms, one-sided-error, query complexity, and testers as follows.

Definition 1: A *testing algorithm* for a property $\mathcal{P}$ is an algorithm that, given query access (by the oracles) to an instance I, accepts every graph from $\mathcal{P}$ with probability at least $2/3$, and rejects every graph that is ϵ-far from $\mathcal{P}$ with probability at least $2/3$. If the testing algorithm accepts every graph from $\mathcal{P}$ with probability 1, then the algorithm is said to be a *one-sided error*. □

The success probability $2/3$ may look too small to apply to actual situations. However, this is not essential, since we can decrease this probability to be any small positive value by iterating the algorithm a constant number of times that depends on the value.

Definition 2 (query complexity and tester): The number of queries made by an algorithm to the given oracle is called the *query complexity* of the algorithm. If the query complexity of a testing algorithm is bounded by a constant independent of the size of instance N (but it may depend on ϵ), then the algorithm is called a *tester*. A property is *testable* if there is a tester for the property.

3. General Tools

3.1 Inequalities Useful to Bound Probabilities

First, we present some general tools useful for analyzing the probabilities of randomized algorithms. Since these tools will be used later in this paper, readers can skip this section and go to the next section for the nonce, and return when they appear later.

Lemma 3: *For any real value x, $1 + x \le e^x$.*

Proof: It is easily obtained by differentiating $e^x - x - 1$ and $e^0 = 1$. □

Theorem 4 (Hoeffding's inequality [19]): *Let $X_1, \ldots, X_s$ be independent random variables bounded as $a_i \le X_i \le b_i$ for all $i \in \{1, \ldots, s\}$. $\overline{X} := \frac{1}{s}\sum_{i=1}^{s} X_i$. Let $\mathrm{Ex}[\overline{X}]$ be the expected value of $\overline{X}$. Then for any $t \ge 0$, the probability that $|\overline{X} - \mathrm{Ex}[\overline{X}]| \ge t$ occurs is bounded by the following inequality:*

$$\Pr[|\overline{X} - \mathrm{Ex}[\overline{X}]| \ge t] \le 2\exp\left(-\frac{2s^2t^2}{\sum_{i=1}^{s}(b_i - a_i)^2}\right). \quad (2)$$

3.2 The Regularity Lemma

Next, we present the monumental lemma known as Szeméredi's regularity lemma. Before explaining this lemma, we need to provide some terms.

For a pair of subsets of vertices $A, B \subseteq V$ of graph $G = (V, E)$, we denote the set of edges between A and B by $E(A, B)$, i.e., $E(A, B) := \{(v, w) \in E \mid v \in A, w \in B\}$. The *density* between A and B is defined as $\mathrm{den}(A, B) := \frac{|E(A,B)|}{|A||B|}$.

Definition 5 (ϵ-regular pair): Let $0 < \epsilon \le 1$ be a real number and $A, B \subseteq V$. A pair (A, B) is called *ϵ-regular* if $|\mathrm{den}(A, B) - \mathrm{den}(X, Y)| \le \epsilon$ for any two subsets $X \subseteq A$ and $Y \subseteq B$ satisfying $|X| \ge \epsilon|A|$ and $|Y| \ge \epsilon|B|$.

Definition 6 (ϵ-regular equipartition): A family of subsets $\mathcal{V} = \{V_1, \ldots, V_k\}$ ($V_i \subseteq V$, $\forall i \in \{1, \ldots, k\}$) is called a *partition* of V if $V_i \cap V_j = \emptyset$ for all $1 \le i < j \le k$ and $V = V_1 \cup \cdots \cup V_k$. This k is called the *order* of the partition. A partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of the vertex set of a graph is called an *equipartition* if $|V_i|$ and $|V_j|$ differ by no more than 1 for all $1 \le i < j \le k$. An equipartition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of the vertex set of a graph is called *ϵ-regular* if all but at most ϵk^2 of the pairs (V_i, V_j) $(i, j \in \{1, \ldots, k\})$ are ϵ-regular. □

Now we can explain the lemma.

Theorem 7 (Szeméredi's regularity lemma [3], [35]): *For every pair of an integer t and a real number $\epsilon > 0$, there exists an integer $T = T_7(t, \epsilon)$ such that any graph with $n \ge T$ vertices has an ϵ-regular equipartition of order k, where $t \le k \le T$.*

3.3 Yao's Minimax Principle

In this subsection, we introduce Yao's minimax principle, which is based on the idea that any randomized algorithm can be regarded as a distribution over deterministic algorithms. We say that a deterministic algorithm A *errs in testing* a property $\mathcal{P}$ on an instance I if A rejects I if $I \in \mathcal{P}$ and A accepts I if I is ϵ-far from $\mathcal{P}$. We also consider a distribution of instances $\mathcal{I}$. The subdistribution consisting of instances whose size is N is denoted by $\mathcal{I}_N$. Clearly $\mathcal{I} = \bigcup_{i=1}^{\infty} \mathcal{I}_i$.

Yao's minimax principle can be expressed in many different forms. The following is one in the property testing form.

Theorem 8 (Yao's minimax principle [8], [16], [36]): *Let $\mathcal{P}$ be a property and $q : \mathbb{Z}^+ \times \mathbb{R}^+ \to \mathbb{Z}^+$ be a function. Assume that for any $\epsilon > 0$ and for infinitely many $N \in \mathbb{Z}^+$, there exists a distribution $\mathcal{I}_N$ such that for every deterministic algorithm A whose query complexity is $q(N, \epsilon)$, the following holds:*

$$\Pr_{I \sim \mathcal{I}_N}[A \text{ errs in testing } \mathcal{P} \text{ on } I] > \frac{1}{3},$$

where $I \sim \mathcal{I}_N$ means that I is chosen according to distribution $\mathcal{I}_N$. Then the query complexity of any algorithm to test $\mathcal{P}$ on parameter ϵ and size N is more than $q(N, \epsilon)$.

Proof: Let $\mathcal{A}$ be an arbitrary randomized algorithm for testing $\mathcal{P}$ with query complexity at most $q(N, \epsilon)$. $\mathcal{A}$ is regarded

as a distribution over deterministic algorithms. Thus the probability that $\mathcal{A}$ errs in testing $\mathcal{P}$ when instances are given over the distribution $\mathcal{I}_N$ is expressed as follows.

$$\begin{aligned}&\Pr_{A\sim\mathcal{A},\ I\sim\mathcal{I}_N}[A \text{ errs in testing } \mathcal{P}]\\ &\geq \min_{A\in\mathrm{supp}(\mathcal{A})}\Pr_{I\sim\mathcal{I}_N}[A \text{ errs in testing } \mathcal{P}],\end{aligned} \tag{3}$$

where $\mathrm{supp}(\mathcal{A})$ is the support of $\mathcal{A}$. From the assumption, (3) is greater than $1/3$. Therefore the probability that $\mathcal{A}$ errs in testing $\mathcal{P}$ when instances are given over the distribution $\mathcal{I}_N$ is more than 1/3. □

4. Basic Techniques

4.1 An Elementary Example of Testers

Here we show an elementary (maybe naive) example of testers to help readers to understand how testers work. A function $f : \{1,\ldots,n\} \to \mathbb{R}$ is *linear* if there are real values $a, b \in \mathbb{R}$ such that $f(x) = ax + b$ for all $x \in \{1,\ldots,n\}$[†]. A function f is ϵ-far from linear if for at least ϵn variables $x \in \{1,\ldots,n\}$, $f(x)$ must be changed to make f linear.

We will show a tester for testing linearity. The oracle of this problem, for any $x \in \{1,\ldots,n\}$, returns $f(x)$. In this algorithm, we assume that $\epsilon \leq 1/4$, since if $\epsilon > 1/4$, ϵ-far is also 1/4-far, and thus using 1/4 replaced with $\epsilon > 1/4$ is sufficient. The algorithm is the following.

procedure LINEARITY
begin
01 choose $s = 2\epsilon^{-1}$ values $S = \{x_1, x_2, \ldots, x_s\}$ from $\{1,\ldots,n\}$ independently at random;
02 check whether all points $(x_i, f(x_i))$, $x_i \in S$ are colinear;
03 if they are colinear, then accept the input; otherwise, reject it;
end.

Theorem 9: LINEARITY *is a one-sided-error tester for linearity with query complexity* $O(\epsilon^{-1})$.

Proof: If the input is linear, then it is clearly accepted by the algorithm. Assume that the input is ϵ-far from linear. Let B be one of the minimum sets of integers $i \in \{1,\ldots,n\}$ such that $f(x_i)$ should be changed to make the input linear. From the assumption, $|B| > \epsilon n$. Thus the probability that a randomly chosen i is not in B is $1 - \epsilon$.

Let L be the line formed by all the points that is not in B. If S is colinear, then (i) $S \cap B = \emptyset$ or (ii) $|S - B| \leq 1$. (Note that (ii) occurs only when the points in $S \cap B$ are accidentally on a line, say L_B, and note that L_B may cross L.) The algorithm accepts the input by mistake in only these cases. The probability that (i) occurs is

$$(1-\epsilon)^s \leq (e^{-\epsilon})^s = e^{-\epsilon s} = e^{-\epsilon(2\epsilon^{-1})} = e^{-2} \leq \frac{1}{6}. \tag{4}$$

[†]In many articles of property testing, "linearity" is used in a different form. To give priority to a good understanding of persons who are not familiar with this area, we use this definition.

The first inequality is obtained from Lemma 3.

The probability that (ii) occurs is at most

$$\begin{aligned}&s\epsilon^{s-1} = 2\epsilon^{s-2} = 2(1-(1-\epsilon))^{s-2}\\ &\leq 2\left(e^{-(1-\epsilon)}\right)^{s-2} = 2\left(e^{-1}\right)^{(1-\epsilon)(s-2)}\end{aligned} \tag{5}$$

The first inequality is obtained from Lemma 3. From $1-\epsilon \geq 3/4$ and $s = 2\epsilon^{-1} \geq 8$ (since $\epsilon \leq 1/4$),

$$(1-\epsilon)(s-2) \geq 3. \tag{6}$$

From (5) and (6), it follows that the probability that (ii) occurs is at most

$$2e^{-3} < \frac{1}{6}. \tag{7}$$

From (4) and (7), the probability that the algorithm accepts the input in mistake is at most $1/6 + 1/6 = 1/3$, i.e., the algorithm rejects it with probability at least $2/3$.

This algorithm is clearly a one-sided-error, since no linear input is ever rejected. The query complexity is $O(s) = O(\epsilon^{-1})$. □

4.2 Testing Triangle-Freeness on Dense Graphs

In this subsection, we show an example of graph-property testing. For an integer $n \in \mathbb{Z}^+$, we denote by K_n a complete n-graph. If a graph does not contain any K_3 as a subgraph, then it is said to be *triangle-free*. Triangle-freeness is clearly a property since any graph isomorphic to a triangle-free graph is also triangle-free.

We first consider a tester for triangle-freeness in the dense-graph model, where the edge-oracle is used. We show a one-sided-error tester of triangle-freeness as follows, where $G = (V, E)$ is a given n-graph and s_ϵ is an integer that is fixed by ϵ and will be defined later.

procedure TRIANGLE-FREENESS
begin
01 choose $s = s_\epsilon$ vertices $S = \{v_1, v_2, \ldots, v_s\}$ from V independently at random;
02 make the subgraph $G(S)$ induced by S through the oracle;
03 if $G(S)$ contains no K_3, then accept the input; otherwise, reject it;
end.

This algorithm is a one-sided-error, since it rejects an input only if it finds a copy of K_3. Showing that it rejects every graph that is ϵ-far from triangle-free with probability at least $2/3$ is not simple.

Lemma 10 ([2]): *For any $\epsilon > 0$, there is an integer $s = s_{10}(\epsilon)$ such that for any graph, if it is ϵ-far from triangle-free, then a subgraph induced by s vertices chosen uniformly at random from the graph contains a K_3 with probability at least $2/3$.*

Proof sketch: Let $G = (V, E)$ be an n-vertex graph ϵ-far

from triangle-free. Let S be the set of vertices chosen by the above algorithm. From Theorem 7, it can be proven that there are $T = T_{10}(\epsilon)$, $\gamma = \gamma_{10}(\epsilon)$ and $t = t_{10}(\epsilon)$ that satisfy the following property: If $n \geq T$, then G has a γ-regular equipartition $\mathcal{V}$ of V with $t \leq |\mathcal{V}| \leq T$. From that G is ϵ-far from triangle-free, (the detail is omitted but) it can be shown that there must be $W_1, W_2, W_3 \in \mathcal{V}$ such that (W_i, W_j) are γ-regular and $\mathrm{den}(W_i, W_j) \geq 2\gamma$ for all $1 \leq i < j \leq 3$. Since $\mathcal{V}$ is an equipartition, $|W_i|/n > 1/2T$ holds. From this, (the detail is omitted again but) it follows that if s is large enough, S includes three vertices $v_1 \in W_1$, $v_2 \in W_2$, and $v_3 \in W_3$ such that $(v_1, v_2), (v_2, v_3), (v_3, v_1) \in E$ with high probability. □

Theorem 11 ([2]): *Triangle-freeness on the dense-graph model is testable by* Triangle-Freeness *with a one-sided-error.*

Proof: We adopt $s_{10}(\epsilon)$ in Lemma 10 as s_ϵ in the procedure. If the graph is triangle-free, the algorithm clearly accept it, and thus the algorithm is a one-sided-error. Assume that the input graph is ϵ-far from triangle-free. From Lemma 10, $G(S)$ contains at least one K_3 with probability at least $2/3$, and the input is rejected with probability at least $2/3$. □

Note that the query complexity of this algorithm is very huge, since the constant $s_{10}(\epsilon)$ in Lemma 10 is a tower of ϵ^{-1}.

4.3 Parameter Testing

In this subsection, we explain a method for approximating a value in constant time. Such a framework is sometimes called *parameter testing*.

Definition 12: Let $x^* \in \mathbb{R}_0^+$ be a nonnegative real value. For a pair of nonnegative real values $\alpha \geq 1$ and $\beta \geq 0$, a value x is said to be an (α, β)-*approximation* of x^* if

$$\frac{x^*}{\alpha} - \beta \leq x \leq \alpha x^* + \beta. \tag{8}$$

□

For an example, we show a constant-time $(1, \epsilon n)$-approximation algorithm for evaluating the number of edges of a given graph in the "bounded-degree-graph model." The *bounded-degree-graph model* (or the *bounded-degree model*, for short) only considers graphs such that every vertex has at most a constant number of neighbours, which is defined as follows.

Definition 13 (degree and bounded-degree): We call the number of adjacent vertices of a vertex $v \in V$ of $G = (V, E)$ the *degree* of v, which is denoted by $\deg_G(v)$, i.e., $\deg_G(v) := |\{w \in V \mid (v, w) \in E\}|$. The subscript G may be omitted if it is clear. For a positive integer $d \in \mathbb{Z}^+$, if $\deg_G(v) \leq d$ for every vertex $v \in V$ in graph $G = (V, E)$, G is said to be a d-*bounded-degree*. The set of d-bounded-degree graphs is denoted by $\Gamma(d)$. Sometimes the d-bounded-degree is called a *bounded-degree* for short. □

The bounded-degree model considers only $\Gamma(d)$, where d is arbitrary. For any graph $G = (V, E) \in \Gamma(d)$, $|E| \leq dn/2$, where $n = |V|$, i.e., $|E| = O(n)$ for any constant d. Hence G is sparse. This means that the edge-oracle is useless, since for almost all queries, the answers will be "there is no edge between the pair of vertices." Thus in the bounded-degree model, the following oracles are used.

- *Degree-oracle*: If an algorithm gives a vertex $v \in V$, this oracle replies $\deg(v)$.
- *Adjacent-vertex-oracle*: If an algorithm gives a pair of $v \in V$ and an integer $i \in \{1, \ldots, \deg(v)\}$, this oracle answers the ith neighbour of v if exists; otherwise, 0.

In this model, the denominator of the distance (Equation(1)) is dn.

Lemma 14: *In the d-bounded-degree model, for any $\epsilon > 0$ and any $0 < p < 1$, a $(1, \epsilon n)$-approximation of $|E|$ can be obtained with probability at least $1 - p$ and query complexity $O(d^2 \epsilon^{-2} \log p^{-1})$.*

Clearly $|E|$ can be expressed by the following equations, where $\mathrm{Ex}[\deg]$ is the average degree of G:

$$|E| = \frac{1}{2} \sum_{v \in V} \deg(v) = \frac{n \cdot \mathrm{Ex}[\deg]}{2}. \tag{9}$$

By using this equation, we can construct an algorithm for estimating $|E|$ as follows.

procedure Graph-Size-Estimation
begin
01 choose $s = \frac{d^2}{8\epsilon^2} \ln \frac{2}{p}$ vertices $S = \{v_1, \ldots, v_s\}$ from V independently at random;
02 calculate $\overline{\deg} = \frac{1}{s} \sum_{i=1}^{s} \deg(v_i)$ and $\overline{m} = n \cdot \overline{\deg}/2$;
03 **output** $\overline{m}$;
end.

Proof of Lemma 14: From Hoeffding's inequality (Theorem 4),

$$\begin{aligned} &\Pr[|\overline{\deg} - \mathrm{Ex}[\deg]| \geq 2\epsilon] \\ &\leq 2 \exp\left(-\frac{2s^2(2\epsilon)^2}{sd^2}\right) = 2 \exp\left(\ln \frac{p}{2}\right) = p. \end{aligned} \tag{10}$$

From $\overline{m} = n \cdot \overline{\deg}/2$ and $|E| = n \cdot \mathrm{Ex}[\deg]/2$, it follows that the above probability is equal to $\Pr[|\overline{m} - |E|| \geq \epsilon n]$. Therefore $\overline{m}$ is a $(1, \epsilon n)$-approximation of $|E|$. □

4.4 Lower Bounds on Query Complexity

Some properties have been known to be non-testable. In this subsection we show how to prove non-testability by using examples.

A graph $G = (V, E)$ is called *bipartite* if V can be partitioned into two subsets V_1 and V_2 such that every edge

is between V_1 and V_2, i.e., $E = E(V_1, V_2)$. Bipartiteness is clearly a property. Bipartiteness is the property that was first found to have a super-constant lower bound for testing. This was obtained by Goldreich and Ron [18].

Theorem 15 ([18]): *For the 3-bounded-degree (graph) model, there is an $\epsilon > 0$ such that any testing algorithm for bipartite with parameter ϵ requires $\Omega(\sqrt{n})$ queries, where n is the number of vertices.*

Proof sketch: In order to use Yao's minimax principle (Theorem 8), we construct a distribution $\mathcal{G}$ on 3-bounded-degree graphs such that any deterministic algorithm whose query complexity less than $\sqrt{n}/4$ cannot distinguish the given graph being bipartite from being 0.01-far from bipartite with probability at least $2/3$.

$\mathcal{G}$ consists of two subsets $\mathcal{G}_1$ and $\mathcal{G}_2$: the former consists of bipartite graphs and the latter consists of graphs that are 0.01-far from bipartite. A graph is given from $\mathcal{G}_1$ or $\mathcal{G}_2$ in the same probability, and any algorithm whose query complexity is small cannot distinguish from which subset the graph comes with high probability. We restrict n as even.

1. $\mathcal{G}_1$ consists of all 3-regular graphs that are composed of a Hamiltonian cycle and a perfect matching.
2. $\mathcal{G}_2$ consists of all 3-regular graphs that are composed of a Hamiltonian cycle and the perfect matching satisfying the following restriction: the distance on the cycle between every two vertices that are connected by a perfect matching edge must be odd.

Clearly all graphs in $\mathcal{G}_2$ are bipartite. It can be also proven that almost all graphs in $\mathcal{G}_1$ are far from bipartite.

Furthermore, it can be proven that any testing algorithm that performs $o(\sqrt{n})$ queries cannot distinguish between a graph chosen randomly from $\mathcal{G}_1$ and a graph chosen randomly from $\mathcal{G}_2$. □

In the same paper [18], it is also shown that testing expander requires $\Omega(\sqrt{n})$ queries.

Stricter $\Omega(n)$ lower bounds on query complexity have been known for some properties. Bogdanov, Obata, and Trevisan [10] showed the first $\Omega(n)$ lower bound for Bounded-degree graph 3-colorability, and furthermore for Vertex Cover, Max Cut, Max 2SAT, Max E3SAT[†], and Max E3LIN-2[††], where all the above problems are in the bounded-degree model, by introducing a reduction method. Later, Yoshida and Ito [39] showed that 3-edge-colorability, Directed/undirected Hamiltonian path/cycle, 3-dimensional matching, and Schaefer-type generalized 3SAT, all in the bounded-degree model, also have the same (linear) lower bounds by introducing some new reduction methods.

[†]EkSAT is SAT, with each clause having exactly k literals.

[††]EkLIN-h is the problem of deciding the satisfiability of a system of linear equations modulo h, with each equation having exactly k variables.

5. Characterizations of Testable Properties

One of the most attractive themes in the area of property testing is finding a combinatorial characterization of testable properties.

5.1 Dense Graphs

For the dense-graph model, a complete combinatorial characterization of testable properties was found by Alon et al. [3]. To put it briefly, this characterization is expressed by the regularity lemma (**3.2**). To explain this a little more, the characterization is said to be "regular reducible," which is shown as follows.

Definition 16 (regularity-instance): A *regularity-instance* R is given by an error-parameter $\epsilon > 0$, an integer k, a set of $\binom{k}{2}$ densities $0 \le \eta_{i,j} \le 1$ indexed by $1 \le i < j \le k$, and a set $\overline{R}$ of pairs (i, j) of a size at most ϵk^2. A graph is said to satisfy the regularity-instance if it has an equipartition $\{V_i \mid 1 \le i \le k\}$ such that for all $(i, j) \notin \overline{R}$ the pair (V_i, V_j) is ϵ-regular and satisfies $|E(V_i, V_j)| = \eta_{i,j}|V_i||V_j|$, i.e., $\mathrm{den}(V_i, V_j) = \eta_{i,j}$. The *complexity* of the regularity instance is $\max(k, 1/\epsilon)$.

Definition 17 (regular-reducible): A graph property $\mathcal{P}$ is *regular-reducible* if for any $\delta > 0$ there exists $r = r_{\mathcal{P}}(\delta)$ such that for any n there is a family $\mathcal{R}$ of at most r regularity-instances each of complexity at most r, such that the following holds for every $\epsilon > 0$ and every n-graph G:

1. If $G \in \mathcal{P}$, then for some $R \in \mathcal{R}$, G is δ-close to R.
2. If G is ϵ-far from $\mathcal{P}$, then for any $R \in \mathcal{R}$, G is $(\epsilon - \delta)$-far[†††] from R.

Theorem 18 ([3]): *For the dense-graph model, a graph property is testable if and only if it is regular-reducible.*

For example, the triangle-freeness considered in 4.2 is regular reducible.

5.2 Bounded-Degree Graphs

We have not obtained a complete characterization of testable properties in the bounded-degree (graph) model. However, an important sufficient condition called "hyperfiniteness" was found.

Definition 19 (hyperfinite): Let $\epsilon > 0$, $t > 0$, and $d > 0$. Let $G = (V, E)$ be an d-degree-bounded n-graph. If one can remove at most ϵdn edges from G so that each connected component of the resulting graph has at most t vertices, then G is called (ϵ, t)-*hyperfinite* (with respect to degree bound d). For a function $\rho : \mathbb{R}^+ \to \mathbb{R}^+$, if G is $(\epsilon, \rho(\epsilon))$-hyperfinite for every $\epsilon > 0$, then G is called ρ-*hyperfinite*. A set $\mathcal{G}$ of

[†††]Here we extend the term "ϵ-far" to negative ϵ, since for every instance can be regarded as ϵ-far (from any family of instances) for any $\epsilon \le 0$ from the definition.

d-degree-bounded graphs is called *ρ-hyperfinite* if $\forall G \in \mathcal{G}$ is ρ-hyperfinite. $\mathcal{G}$ is called *hyperfinite* if there is a function ρ such that $\mathcal{G}$ is ρ-hyperfinite. □

Note that if a graph is hyperfinite, then it is close to a graph that can be partitioned into small connected components. If a graph can be partitioned into small, connected components, then a local search from randomly chosen vertices is well suited. In fact, the following theorem was given by Newman and Sohler [33].

Theorem 20 ([33]): *In the bounded-degree model, every graph property is testable for any hyperfinite family of graphs.*

Unfortunately this theorem is not a necessary condition, e.g., k-edge/vertex-connected for any fixed $k \geq 3$ is not hyperfinite but testable [18], [38]. However, a necessary condition based on hyperfiniteness was also found by Fichtenberger et al. [13] as shown in the following.

A *subproperty* of a property $\mathcal{P}$ is a property that is a subset of $\mathcal{P}$. A property is *non-trivially testable* if it is testable and there exists $\epsilon > 0$ such that there is an infinite number of graphs that are ϵ-far from the property.

Theorem 21 ([13]): *Every testable property of bounded-degree graphs is either finite or contains an infinite hyperfinite subproperty. Also, the complement of every non-trivially testable graph property contains an infinite hyperfinite subproperty.*

Ordinarily we suppose that testing algorithms know the size of the input, e.g., the number of vertices of a given graph. Alon and Shapira [5], however, introduced an idea of the *oblivious* tester, which must work without knowing the size, and showed a complete characterization of one-sided-error oblivious testers.

Recently, combinatorial characterizations of one-sided error testability for monotone and hereditary properties in the bounded-degree model were presented by Ito, Khoury, and Newman [22]. If for any $G \in \mathcal{P}$, a graph obtained by removing an arbitrary edge (resp., vertex) is also in $\mathcal{P}$, then $\mathcal{P}$ is called *monotone* (resp., *hereditary*) [4]. Note that any minor-closed property [12] (including planar) and k-colorability for any $k \in \mathbb{Z}^+$ (including bipartite) are monotone and hereditary. This characterization covers not only undirected graphs but also digraphs. In the paper, an idea of "forbidden configurations" was presented. This idea may be useful in obtaining a characterization of one-sided-error testable properties with no restriction on the bounded-degree model.

5.3 General Graphs

We have shown two models on graphs: the dense-graph model and the bounded-degree model. Another popular model is the general (graph) model. This model is a common generalization of the other two models.

In this model, an upper bound d of the *average* degree of every graph is given, i.e., $d \geq \sum_{i \in V} \deg(i)/n$ for every graph. Ordinarily we assume that d is a constant, and thus this model treats sparse graphs. The distance is defined as Eq. (1), where d is the upper bound on the average degree. This model allows all oracles that can be used in the other two models.

As written above, this model is a common generalization of the other two models and it is inevitably more difficult than them. Although we are now far from getting a characterization, class $\mathcal{HSF}$ (Hierarchal Scale Free), which models complex networks was presented by Ito [20]. It is shown that any property is testable in $\mathcal{HSF}$, i.e., a universal tester[†] is given in this class.

In the general-graph model, while no other universal (constant-time) tester has been known, universal testing algorithms with polylog(n)-time query complexity have been found on forests [28] and outerplanar graphs [7].

5.4 Other Results

We briefly introduce some other results on not only characterizations, but also various types of problems. On affine-invariant functions, Yoshida [37] presented a complete characterization of testable properties.

Batu, Berenbring, and Sohler [6] gave a parameter-testing algorithm for the bin packing problem with query complexity $\tilde{O}(\sqrt{n} \cdot \mathrm{poly}(\epsilon^{-1}))$. Ito, Kiyoshima, and Yoshida [23] showed a parameter-testing algorithm for the knapsack problem with query complexity $\tilde{O}(\epsilon^{-4})$. For EXPTIME-complete problems, the generalized chess, Shogi (Japanese chess), and Xiangqi (Chinese chess) were proven to be all testable by Ito, Nagao, and Park [24]. Ito and Ueda applied sublinear-time algorithms to the cake-cutting problem [25].

Lovász and Vesztergombi [30] introduced an idea of *nondeterministic* property testing, and showed a relation to deterministic (i.e., ordinary) property testing. In response to this paper Gishboliner and Shapira [14] provideed an additional results using Szemerédi's regularity lemma.

6. Sublinear Progressive Algorithm

6.1 What is a Sublinear Progressive Algorithm

In this section, we present the idea of "sublinear progressive algorithms." Constant- or sublinear-time algorithms are very useful when we need to get a solution quickly if a problem suddenly occurs and we should need to read very huge data if we used a traditional polynomial-time algorithm. In such a case, it is useful to get an approximate solution quickly by using a constant- or sublinear-time algorithm. Even in such a case, however, after getting a temporary solution, we may hope to get a better solution gradually as time permits. Ideally, we want to get better and better solutions gradually,

[†]A *universal tester* means that it can test every property in a wide class of instances (e.g., graphs). Since this "wide" is intuitive idea, "a universal tester" is not a mathematically defined term.

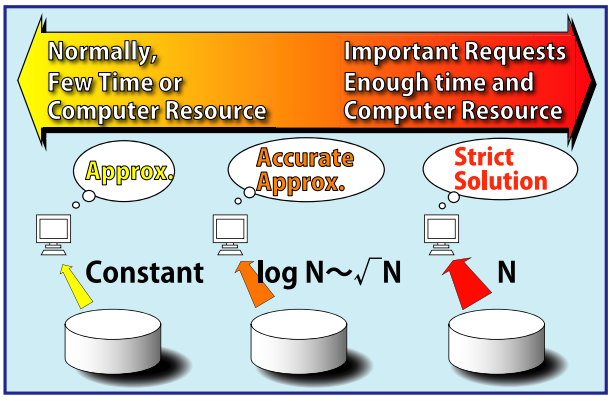

Fig. 2 Progressive algorithms.

and to get an exact solution finally in the same time as if we applied an exact algorithm from the biginning.

We call such algorithms "sublinear progressive algorithms," which are illustrated in Fig. 2. Only preliminary results on this idea have been presented by the authors in an international conference [11][†].

In this section, we show that for any problem, if there are both a constant-time algorithm and an exact algorithm, we can construct an ideal sublinear-time algorithm, i.e., given any positive integers $r \geq 2$ and $t \geq 1$, an algorithm that outputs r solutions $S_1, S_2, \ldots, S_r$ in this order, where S_r is the exact solution, under the constraint that

(1) S_1 is obtained in time $O(t)$,
(2) for any $i \in \{1, \ldots, r\}$ the worst-case computation time T_i for getting S_i is $O(T^*(S_i))$, where $T^*(S_i)$ is the computation time to get S_i by using the original constant-time algorithm (if $i \in \{1, \ldots, r-1\}$) or exact algorithm (if $i = r$), and
(3) $R := \max_{i \in \{2,\ldots,r\}} T_i/T_{i-1}$ is minimized in the big-O sense.

The meaning of Constraint (3) is that the time we should wait for the next solution follows a geometric progression. If it follows an arithmetic progression, then $T_2 \approx T_r/r$. T_r/r is very close to T_r, which is the time to output the final (exact) solution, in the big-O sense. Thus, in this case, we must wait for a very long time (until T_r/r) to get the second solution if $T^*(S_r)$ is very huge. By introducing the geometric progression, the waiting times become gradually longer and longer. This is reasonable in actual situations.

6.2 Related Problems

The term "progressive algorithms" is already known in the context of polynomial-time algorithms. In this context, some studies have discussed the following: Convex hulls and computing popular places from trajectory data [1], group Steiner tree search [29], sorting in the external memory model [31], Euclidean minimum spanning tree [32] have been developed.

An analogous concept to a progressive algorithm is an anytime algorithm [9]. An *anytime algorithm* is an algorithm that returns some answer for any allocation of computation time, and is expected to return better answers when given more time. However, the performance of anytime algorithms has been evaluated experimentally.

6.3 Sublinear Progressive Algorithm Theorem

We present *Sublinear Progressive Algorithm Theorem* (*SPA Theorem*, for short) as follows. While we show only the property-testing version here, the parameter-testing version is easily defined similarly and is omitted.

Definition 22: Let $\mathcal{P}$ be a property. For positive real numbers $\epsilon > 0$ and $0 < p < 1/2$, a random valuable $S \in \{\text{yes}, \text{no}\}$, which is the output of testing algorithm for $\mathcal{P}$, is said to be an $(\epsilon, 1-p)$-*solution* if the input is in $\mathcal{P}$, then $S = \text{yes}$ with probability at least $1-p$ and if the input is ϵ-far from $\mathcal{P}$, then $S = \text{no}$ with probability at least $1-p$. □

Our algorithm, given $r \geq 2$, outputs r solutions $S_1, S_2, \ldots, S_r$, where for $i = 1, \ldots, r-1$, S_i is an $(\epsilon_i, 1-p_i)$-solution, and S_r is the exact solution. Let T_i be the time calculated to get S_i by the progressive algorithm. Constraint (2) in 6.1 means that every intermediate solution and the final solution must be calculated in almost the same time as the original constant-time or exact algorithms. In order to satisfy Constraint (3), these solutions are obtained in the time following an arithmetic progression, i.e., $T_i = O(t\tau^{i-1})$ for some $\tau > 1$. Moreover, from the requirement of actual applications, ϵ_i and p_i should become better and better gradually.

The following theorem shows that for any property, if it has a constant-time algorithm and an exact algorithm (an exponential-time algorithm is allowed), then a sublinear progressive algorithm satisfying the above conditions exists.

Theorem 23 (SPA Theorem: property testing version): *Let $\mathcal{P}$ be a property. Suppose that there exist a tester* Alg_0 *whose time complexity is $T^*(\epsilon, 1-p)$, where $\epsilon > 0$ is an approximation parameter and $0 < p \leq 1/2$ is a failure probability, and an exact algorithm* Alg_1 *whose time complexity is $T^*(n)$, where n is the size of the input, for $\mathcal{P}$. Then there exists an algorithm, given any positive integers $r \geq 1$ and $0 < t \leq T^*(n)$, that provides r solutions $S_1, S_2, \ldots, S_r$, in time $T_1, T_2, \ldots, T_r$, respectively, and satisfies the following conditions:*

(1) $T_1 = \Theta(t)$.
(2.1) *S_r is the exact solution and S_i is an $(\epsilon_i, 1-p_i)$-solution for some $\epsilon_i > 0$ and $0 < p_i \leq 1/2$, $\forall i \in \{1, \ldots, r-1\}$.*
(2.2) *$T_r = O(T^*(n))$ and $T_i = O(T^*(\epsilon_i, 1-p_i))$, $\forall i \in \{1, \ldots, r-1\}$.*
(3) *For $i \in \{1, \ldots, r-1\}$, $T_i = O(t\tau^{i-1})$ for some $\tau \geq 2$, and moreover if $\tau > 2$, then $T_i = \Theta(t\tau^{i-1})$.*
(4) *Both ϵ_i and p_i are decreasing functions of i.*

Note that in $O(*)$ and $\Theta(*)$ in this theorem, ϵ, r, and t are regarded as variables. Although we did not define ϵ_r and p_r in this theorem, we can introduce them as $\epsilon_r = 0$ and $p_r = 0$ (since S_i is the exact solution).

[†]Oral presentation only. No proceedings published.

In Condition (3) of this theorem, the common ratio of increasing T_i is at least 2, i.e., $\tau \geq 2$. This is because if τ is smaller than 2, then we do not need to calculate solutions so frequently, and it is sufficient to use 2 as the common ratio. Moreover, the latter half of Condition (3) "if $\tau > 2$, then $T_i = \Theta(t\tau^{i-1})$" is important. Since if this constraint does not exist, then $T_i = O(t\tau^{i-1})$ (for $i \in \{1, \ldots, r-1\}$) can be trivially satisfied by letting τ be very huge, e.g., $\tau \geq T^*(n)/T_1$.

Proof of Theorem 23: All of the Conditions (*) appearing in this proof represent the conditions in the statement of this theorem.

From Condition (1), we calculate ϵ_1 and p_1 that satisfy $T^*(\epsilon_1, 1-p_1) \leq \max\{2t, t+c\}$ for an appropriate constant $c > 0$, which is required to output $S_1 \in \{\text{yes}, \text{no}\}$ when t is too small. Note that since any random solution is an $(\epsilon, 1/2)$-solution for any ϵ, there must be such ϵ_1 and p_1.

$$T_1 := \max\{2t, t+c\}. \tag{11}$$

Let τ and $T'_1, \ldots, T'_r$ be as follows, where n is the size of the input.

$$\tau = \max\left\{\left(\frac{T^*(n)}{T_1}\right)^{\frac{1}{r-1}}, 2\right\}, \tag{12}$$

$$T'_i := \min\{T_1\tau^{r-1}, T^*(n)\} \quad (i = 1, \ldots, r). \tag{13}$$

If $(T^*(n)/T_1)^{1/(r-1)} < 2$ in (12), then $\tau = 2$ and T'_i may reach $T^*(n)$ before $i = r$, and the algorithm can stop there, i.e., if $\exists k, T'_{k-1}\tau \geq T^*(n)$, then we stop the increment of T'_i at this point, i.e., $T'_k = T'_{k+1} = \cdots = T'_r$. Let k be the minimum positive integer that satisfies the above condition. Note that $k < r$ occurs only if $\tau = 2$. From these definitions, $T'_1 = T_1$ and $T'_k = T'_r = T^*(n)$.

For each $i = 2, \ldots, k-1$, we calculate ϵ_i and p_i that satisfy $T^*(\epsilon_i, 1-p_i) = \Theta(T'_i)$. Since $T^*(\epsilon, 1-p)$ is a decreasing function of both ϵ and p and $T^*(\epsilon, 1-p) \leq T^*(n)$ for any $0 \leq \epsilon \leq 1$ and $0 \leq p \leq 1$ ($\because$ Alg_1 outputs $(0,1)$-solution and thus if $T^*(\epsilon, 1-p) > T^*(n)$) for some ϵ and p, then we can replace Alg_0 with Alg_1 for such ϵ and p), there must be such ϵ_i and p_i satisfying $\epsilon_i \leq \epsilon_{i-1}$ and $p_i \leq p_{i-1}$.

Now we present the progressive algorithm. Note that $k < r$ occurs only if $\tau = 2$, otherwise $k = r$.

procedure PROGRESSIVE
begin
01 **do from** $i = 1$ **to** $k-1$;
02 **if** $T'_i = T^*(n)$ **then stop**;
03 calculate ϵ_i and p_i;
04 **call** $\mathrm{Alg}_0(\epsilon_i, 1-p_i)$ and **output** the solution (S_i);
05 **enddo**
06 **call** Alg_1 and **output** the solution (S^*);
end.

We show that this algorithm satisfies the conditions. Conditions (1), (2.1), and (4) are clear from the construction of the algorithm. From $T'_1 = T_1 = \max\{2t, t+c\}$, Condition (2.2) is clear for $i = 1$.

For every $i \in \{2, \ldots, k\}$,

$$\begin{aligned} T_i &= \sum_{j=1}^{i} T'_i = \sum_{j=1}^{i} T'_1\tau^{j-1} = \frac{\tau}{\tau-1}(\tau^{j-1}-1)T'_1 \\ &\leq 2\tau^{j-1}T'_1 \quad (\because\ \tau \geq 2) \\ &= 2T'_i. \end{aligned}$$

From this, it follows that $T_k = T_r = O(T^*(n))$ and $O(T^*(\epsilon_i, 1-p_i))$ for $i \in \{2, \ldots, k-1\}$, i.e., Condition (2.2) is satisfied for all i. The former half of Condition (3) is also clear. If $\tau > 2$, then $k = r$ and the latter half of Condition (3) is also satisfied. □

Theorem 23 assures that we can construct a progressive algorithm without losing any computation time in the big-O sense. Furthermore, it can be observed that the above algorithm is the optimum in some sense as shown below.

Observation 24: *The algorithm constructed in the proof of Theorem 23 is the algorithm that minimizes the ratio* $R := \max_{i \in \{2,\ldots,r\}} T_i/T_{i-1}$ *in the big-O sense, among all algorithms that satisfy all of the conditions.*

Proof of Observation 24: Let $\widehat{T}_i$ be the time when the ith solution S_i is outputted by the optimal algorithm. To obtain the exact solution S_r we need at least $T^*(n)$ time, i.e., $\widehat{T}_r \geq T^*(n)$. Moreover, the first solution S_1 must be outputted in time $\Theta(t)$, since $T_1 = \Theta(t)$ also, we have $\widehat{T}_1 \leq c'T_1$ for some constant $c' \geq 1$. Thus

$$\frac{\widehat{T}_r}{\widehat{T}_1} \geq \frac{T^*(n)}{c'T_1} = \frac{\tau^{r-1}}{c'}. \quad (\because \text{(12) and } \tau \geq 2.)$$

From

$$\widehat{T}_r \leq R^{r-1}\widehat{T}_1,$$

it follows that

$$\begin{aligned} R &\geq \left(\frac{\widehat{T}_r}{c'\widehat{T}_1}\right)^{\frac{1}{r-1}} \geq \left(\frac{1}{c'}\right)^{\frac{1}{r-1}} \tau \geq \frac{\tau}{c'} \quad (\because\ c' \geq 1.) \\ &= \Omega(\tau). \end{aligned}$$

□

6.4 Representative Example of ϵ_i and p_i

We consider a representative case on progressive algorithms. In many constant-time algorithms, the computation time for getting an $(\epsilon, 1-p)$-solution can be represented by

$$t(\epsilon, 1-p) = c'\left(\frac{1}{\epsilon}\right)^{c''} \lg\frac{1}{p}, \tag{14}$$

where c' and c'' are constants that depend on the property.

Under this assumption we show the concrete expressions of ϵ_i and p_i that appeared in Theorem 23. However,

there is a freedom to fix the ratio between ϵ_i and p_i, i.e., even if the value of $t(\epsilon_i, 1-p_i)$ is fixed, there are two variables, ϵ_i and p_i, and we cannot fix them.

In the proof of Theorem 23, computation time of S_i is longer than one of S_{i-1} at the ratio of τ, i.e., $t(\epsilon_i, 1-p_i)/t(\epsilon_{i-1}, 1-p_{i-1}) = \tau$. This τ can be divided into the ϵ-depending time, $(1/\epsilon)^{c''}$, and the p-depending time, $\lg(1/p)$. Here, we fix the ratio as $\rho : 1-\rho$ for arbitrary $0 \le \rho \le 1$ in every step, i.e., the ϵ-depending time becomes τ^{ρ} times longer and the p-depending time becomes $\tau^{1-\rho}$ times longer in each step.

The freedom in fixing ϵ_1 and p_1 still remains. If p_1 is fixed, then from $t(\epsilon_1, 1-p_1) = T_1$ and (14), we obtain

$$\epsilon_1 = \left(\frac{c' \lg \frac{1}{p_1}}{T_1}\right)^{1/c''}. \tag{15}$$

Note that $\epsilon_1 \le 1$. From this, $c' \lg(1/p_1) \le T_1$, and thus

$$p_1 \ge \left(\frac{1}{2}\right)^{T_1/c'}. \tag{16}$$

By setting c in (11) to be larger than or equal to this c', p_1 can be at most $1/2$. We can choose any p_1 that satisfies (16). From the above discussions, ϵ_i and p_i for $i \in \{2, \ldots, r-1\}$ are expressed as follows.

$$\epsilon_i = \frac{\epsilon_1}{\tau^{\rho(i-1)/c}}, \tag{17}$$

$$p_i = {p_1}^{\tau^{(1-\rho)(i-1)}}. \tag{18}$$

7. Concluding Remarks

This paper consisted of two parts. The first part was a survey on the sublinear computation paradigm, mainly on constant-time algorithms. In this part, after presenting basic tools and some examples, we showed key results mainly on the characterization of testing properties.

In the second part, we considered a theory of sublinear progressive algorithms, which is the first publishment in journal papers. We presented the Sublinear Progressive Algorithm Theory (SPA Theory, for short), which enables to make a sublinear progressive algorithm for any property if it has a constant-time algorithm and an exact algorithm (an exponential-time one is allowed) without losing any computation time in the big-O sense.

Due to the space restrictions, however, we were limited to only a few subjects. If you are interested in this area, please refer to the special issues and books on the sublinear-time paradigm [26], [27] and property testing [8], [15], [16], [21], which are helpful.

The 21st Century can be called the Big Data Era. The larger big data becomes, the more we need sublinear- and constant-time algorithms. The importance of this area will become greater and greater. We hope many ambitious young researchers will join us in this endeavor.

Acknowledgments

We would like to express our gratitude to Professor Ilan Newman of the University of Haifa, Israel, Professor Naoki Katoh of the University of Hyogo, Japan, Associate Professor Yuichi Yoshida of the National Institute of Informatics, Japan, and Associate Professor Suguru Tamaki of the University of Hyogo, Japan for their great help. We appreciate Mr. Mikiya Imura, who was a student of Tokyo Institute of Technology, Japan, for his preliminary research on progressive algorithms with us. We also thank the members of the Foundations of Innovative Algorithms for Big Data, who have continuously helped us. We are also grateful to the anonymous referees for their careful and detailed comments, which improved this article much better. This work was partially supported by JST CREST JPMJCR1402, and JSPS KAKENHI 15K11985 and 20K11671.

References

[1] S.P.A. Alewijnse, T.M. Bagautdinov, M. de Berg, Q.W. Bouts, A.P. ten Brink, K. Buchin, and M.A. Westenberg, "Progressive geometric algorithms," Proc. SOCG'14, pp.50–59, 2014 (Journal version: JoCG, vol.6, no.2, pp.72–92, 2015).

[2] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, "Efficient testing of large graphs," Proc. FOCS 99, pp.656–666, 1999 (Journal version: Combinatorica, vol.20, pp.451–476, 2000).

[3] N. Alon, E. Fischer, I. Newman, and A. Shapira, "A combinatorial characterization of the testable graph properties: It's all about regularity," SIAM J. Comput., vol.39, no.1, pp.143–167, 2009.

[4] N. Alon and A. Shapira, "Every monotone graph property is testable," Proc. STOC 2005, pp.128–138, 2005. (Journal version: SIAM J. Comput., vol.38, no.6, pp.1703–1727, 2008).

[5] N. Alon and A. Shapira, "A characterization of the (natural) graph properties testable with a one-sided error," Proc. FOCS 2005, pp.429–438, 2005 (Journal version: SIAM J. Comput., vol.37, no.6, pp.1703–1727, 2008).

[6] T. Batu, P. Berenbrink, and C. Sohler, "A sublinear-time approximation scheme for bin packing," Theor. Comput. Sci., vol.410, no.47–49, pp.5082–5092, 2009.

[7] J. Babu, A. Khoury, and I. Newman, "Every property of outerplanar graphs is testable," Proc. RANDOM 2016, LIPICS, pp.21:1–21:19, 2016.

[8] A. Bhattacharyya and Y. Yoshida, Property Testing — Problems and Techniques, Springer, 2021.

[9] M. Boddy and T.L. Dean, "Solving time-dependent planning problems," Proc. IJCAI'89, vol.2, pp.979–984, 1989.

[10] A. Bogdanov, K. Obata, abd L. Trevisan, "A lower bound for testing 3-colorability in bounded-degree graphs," Proc. FOCS 02, pp.93–102, 2002.

[11] K. Chiba, M. Imura, H. Ito, and I. Newman, "Sublinear progressive algorithms — The framework and fundamental theorems," The 5th International Workshop on Innovative Algorithms for Big Data (IABD2019), Kyoto, Japan, Nov. 2019.

[12] R. Diestel: Graph Theory, 5th ed., Springer, 2016.

[13] H. Fichtenberger, P. Peng, and C. Sohler, "Every testable (infinite) property of bounded-degree graphs contains an infinite hyperfinite subproperty, arXiv: 1811.02937, 2018 (also appeared in SODA2019).

[14] L. Gishboliner and A. Shapira, "Deterministic vs non-deterministic graph property testing," Electronic Colloquium on Computational Complexity, Report no.59, pp.1–17, 2013.

[15] O. Goldreich, ed., Property Testing — Current Research and Surveys,

LNCS 6390, Springer, 2010.
[16] O. Goldreich, Introduction to Property Testing, Cambridge University Press, 2017.
[17] O. Goldreich, S. Goldwasser, and D. Ron, "Property testing and its connection to learning and approximation," J. ACM, vol.45, no.4, pp.653–750, July 1998.
[18] O. Goldreich and D. Ron, "Property testing in bounded degree graphs," Proc. STOC 1997, pp.406–415, 1997 (Journal versioon: Algorithmica, vol.32, no.2, pp.302–343, 2002).
[19] W. Hoeffding, "Probability inequalities for sums of bounded random variables," J. Am. Stat. Assoc., vol.58, no.301, pp.13–30, 1963.
[20] H. Ito, "Every property is testable on a natural class of scale-free multigraphs," Proc. ESA 2016, LIPICS, vol.49, pp.21:2–21:15, 2016 (ISBN 978-3-95977-005-7).
[21] H. Ito, "What graph properties are constant-time testable? — Dense graphs, sparse graphs, and complex networks," The Review of Socionetwork Strategies, vol.13, no.2, pp.101–121, Springer, 2019.
[22] H. Ito, A. Khoury, and I. Newman, "On the characterization of 1-sided error strongly-testable graph properties for bounded-degree graphs," Comput. Complex., vol.29, no.1, pp.1–45, Springer, 2020.
[23] H. Ito, S. Kiyoshima, and Y. Yoshida, "Constant-time approximation algorithms for the knapsack problem," Proc. TAMC 2012, LNCS 7287, pp.131–142, Springer, 2012.
[24] H. Ito, A. Nagao, and T. Park, "Generalized shogi, chess, and xiangqui are constant-time testable," IEICE Trans. Fundamentals, vol.E102-A, no.9, pp.1126–1133, Sept. 2019.
[25] H. Ito and T. Ueda, "How to solve the cake-cutting problem in sublinear time," Proc. 8th International Conference on Fun with Algorithms (FUN2016), LIPICS, vol.49, pp.21:1–21:15, 2016 (ISBN 978-3-95977-005-7).
[26] N. Katoh, et al., eds.: "Special Issue on Foundations of Innovative Algorithms for Big Data — Sublinear Computational Paradigm and Its Expansions", The Review of Socionetwork Strategies, vol.13, no.2, 2019.
[27] N. Katoh, et al. eds., Sublinear Computation Paradigm — Algorithmic Revolution in the Big Data Era, Springer, 2022 (Open Access: DOI: 10.1007/978-981-16-4095-7).
[28] M. Kusumoto and Y. Yoshida, "Testing forest-isomorphizm in the adjacency list model," Proc. of ICALP2014 (1), LNCS 8572, pp.763–774, 2014.
[29] R.-H. Li, L. Qin, J. Xu Yu, and R. Mao, "Efficient and progressive group steiner tree search," Proc. SIGMOD/PODS'16, pp.91–106, 2016.
[30] L. Lovász and K. Vesztergombi, "Non-deterministic graph property testing," Combinatorics, Probability and Computing, vol.22, no.5, pp.749–762, 2013.
[31] A. Mesrikhani and M. Farshi, "Progressive sorting in the external memory model," The CSI Journal on Computer Science and Engineering, vol.15, no.2, pp.1–4, 2018.
[32] A. Mesrikhani, M. Farshi, and M. Davoodi, "Progressive algorithm for euclidean minimum spanning tree," Proc. ICCG'18, pp.29–32, 2018.
[33] I. Newman and C. Sohler, "Every property of hyperfinite graphs is testable," Proc. STOC 2011, ACM, pp.675–784, 2011 (Journal version: SIAM J. Comput., vol.42, no.3, pp.1095–1112, 2013).
[34] R. Rubinfeld and M. Sudan, "Robust characterizations of polynomials with applications to program testing," SIAM J. Comput., vol.25, no.2, pp.252–271, 1996 (Conference version was appeared in 1992).
[35] E. Szemerédi, "Regular partitions of graphs," J.C. Bermond, J.C. Fournier, M. Las Vergnas, and D. Sotteau, eds., Colloq. Internat. CNRS, CNRS, Paris, pp.399–401, 1978.
[36] A.C. Yao, "Lower bounds to randomized algorithms for graph properties," Proc. FOCS 1987, pp.393–400, 1987.
[37] Y. Yoshida, "A characterization of locally testable affine-invariant properties via decomposition theorems," Proc. STOC 14, pp.154–163, 2014.
[38] Y. Yoshida and H. Ito, "Property testing on k-vertex-connectivity of graphs," Proc. ICALP 08, Part I, LNCS 5125, pp.539–550, Springer, 2008 (Journal version: Algorithmica, vol.62, no.3–4, pp.701–712, 2012).
[39] Y. Yoshida and H. Ito, "Query-number preserving reductions and linear lower bounds for testing," IEICE Trans. Inf. & Syst., vol.E93-D, no.2, pp.233–240, Feb. 2010.

Kyohei Chiba received a B.E. degree from the Department of Communication Engineering and Informatics at The University of Electro-Communications in 2017, and received an M.E. degree from the Graduate School of Informatics and Engineering at The University of Electro-Communications in 2019. His main research interests are in theoretical computer science, graph theory and discrete algorithms.

Hiro Ito received B.E., M.E., and Ph.D. degrees from the Department of Applied Mathematics and Physics, the Faculty of Engineering, Kyoto University in 1985, 1987, and 1995, respectively. In 1987–1996, 1996–2001, and 2001–2012, he was a member of NTT Laboratories, Toyohashi University of Technology, and Kyoto University, respectively. Since 2012, he has been a full professor in the School of Informatics and Engineering at The University of Electro-Communications (UEC). He has been engaged in research on discrete algorithms mainly on graphs and networks, discrete mathematics, recreational mathematics, and algorithms for big data. Dr. Ito is a member of IEICE, the Operations Research Society of Japan, the Information Processing Society of Japan, and the European Association for Theoretical Computer Science. He is also a member of the steering committee of JCDCG^3.