Chapter 1
What Is the Sublinear Computation Paradigm?

Naoki Katoh and Hiro Ito

Abstract This chapter introduces the "sublinear computation paradigm." A sublinear-time algorithm is an algorithm that runs in time sublinear in the size of the instance (input data). In other words, the running time is $o(n)$, where n is the size of the instance. This century marks the start of the era of big data. In order to manage big data, polynomial-time algorithms, which are considered to be efficient, may sometimes be inadequate because they may require too much time or computational resources. In such cases, sublinear-time algorithms are expected to work well. We call this idea the "sublinear computation paradigm." A research project named "Foundations on Innovative Algorithms for Big Data (ABD)," in which this paradigm is the central concept, was started under the CREST program of the Japan Science and Technology Agency (JST) in October 2014 and concluded in September 2021. This book mainly introduces the results of this project.

1.1 We Are in the Era of Big Data

The twenty-first century can be called the era of Big Data. The number of webpages on the Internet was estimated to be more than 1 trillion ($=10^{12}$) in 2008 [22], and the number of websites grows ten times in these 10 years [21]. Thus the number of webpages is estimated to be more than 10 trillion ($=10^{13}$) now. If we assume that 10^6 bytes($\approx 10^7$ bits) of data is contained in a single webpage on average,[1] then the total amount of the data stored on the Internet would be more than 100 exabits ($=10^{20}$ bits)! The various actions that everyone performs are collected by our smartphones and are stored in the memory of storage devices around the world. The remarkable development of computer memory has made it possible to store this information.

[1] Note that one 1080×1920 pixel digital photo consists of more than 2×10^6 pixels.

N. Katoh
University of Hyogo, 8-2-1 Gakuennishi-machi, Nishi-ku, Kobe, Hyogo 651-2197, Japan
e-mail: naoki.katoh@gsis.u-hyogo.ac.jp

H. Ito (✉)
The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
e-mail: itohiro@uec.ac.jp

N. Katoh et al. (eds.), *Sublinear Computation Paradigm*,
https://doi.org/10.1007/978-981-16-4095-7_1

However, the ability to store data and the ability to make good use of the data are different problems. The speed of the data transfer using IEEE 802.11ac is 6.9 Gbps. Using this, it would take 1.7 days to read 1 petabit (10^{15} bit) of data. To read 1 exabit (10^{18} bit) of data, we would need over 4 years! Although the speed of data transfer is expected to continue to increase, the amount of available data is also expected to grow even faster.

This situation can create new problems that did not arise in past centuries, such as requiring a huge amount of time just to read an entire dataset. We are thus faced with new problems in terms of computation.

1.2 Theory of Computational Complexity and Polynomial-Time Algorithms

In the area of the theory of computational complexity, the term "polynomial-time algorithms" is often as a synonym for "efficient algorithms." A *polynomial-time algorithm* is an algorithm that runs in time expressed by a function polynomial of the size of the instance (i.e., the input). For example, consider the sorting problem that takes a set of positive integers $a_1, \ldots, a_n$ as input and outputs a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_{\pi(i)} \leq a_{\pi(i+1)}$ for every $i \in \{1, \ldots, n-1\}$. In this problem, the input is expressed by n integers and thus the input size is n.[2]

We now briefly introduce the theory of computational complexity. Theoretically, the computation time of an algorithms is expressed in terms of the number of basic units of calculations (i.e., the basic arithmetic operations, reading or writing a value in a cell in memory, and comparison of two values[3]). The complexity is then expressed as a function of n, say $T(n)$, where n is the (data) size of the input. If there exists a fixed integer k such that $T(n) = O(n^k)$, then we say that the algorithm runs in polynomial time.

For example, the sorting problem can be solved in $O(n \log n)$ time, which is polynomial, and it has been proven that this is the minimum in the big-O sense, meaning that no algorithm exists that runs in $o(n \log n)$-time. In contrast, for the *partitioning problem*, which is the problem of finding a subset B of a given set A consisting of n integers $a_1, \ldots, a_n$ such that $\sum_{a_i \in B} a_i = \frac{1}{2} \sum_{a_i \in A} a_i$, no polynomial-time algorithms have been found and the majority of researchers believe that no such algorithm exists.[4]

---

[2] More rigorously, representing an integer a requires around $\log_2 a$ bits. However, in the area of the theory of computational complexity, we usually use the assumption that one integer is stored in one cell (byte) of the memory. Since this assumption may cause some strange results if pathologically huge integers are used, these integers are prohibited.

[3] In order to avoid excessive calculations, we assume that each integer consists of at most $\log_2 n$ bits, where n is the number of integers treated in the instance.

[4] This is equivalent to the well-known "P vs. NP problem," which is one of the seven Millennium Prize open problems in mathematics.

For many problems, constructing an exponential-time algorithm is easy. For the partitioning problem, for example, an algorithm that tests all subsets of A clearly solves the problem, and this requires $2^n \cdot O(n)$ time, which is exponential. Therefore, the existence of an exponential-time algorithm is considered to be trivial for many cases. Constructing polynomial-time algorithms, however, requires additional ideas in many cases.

1.3 Polynomial-Time Algorithms and Sublinear-Time Algorithms

1.3.1 A Brief History of Polynomial-Time Algorithms

The idea that "polynomial-time algorithms are efficient" is sometimes called Cobham's Thesis or Cobham–Edmonds' Thesis, which is named after Alan Cobham and Jack Edmonds [4]. Cobham [3] identified tractable problems with the complexity class P, which is the class of problems solvable in polynomial-time with respect to the input size. Edmonds also stated the same thing in [7].

Although these papers were published in 1965, the idea behind this thesis seems to have been a commonly held belief among researchers in the late in 1950s. For example, Kruskal's algorithm and Prim's algorithms, which are both almost linear-time algorithms for the minimum spanning tree problem, were presented in 1956 [16] and 1957 [17], respectively. Dijkstra's algorithm, which is an almost linear-time algorithm for the shortest path problem with positive edge lengths, was presented in 1959 [6]. Ford and Fulkerson presented the augmenting path algorithm for the maximum flow problem in 1956 [8]. The blossom algorithm was proposed by Jack Edmonds in 1961 for the maximum matching problem on general (i.e., not necessarily bipartite) graphs [7].

In 1971, Cook proposed the idea of NP-completeness and proved that the satisfiability problem (SAT) is NP-complete [5]. NP-complete problems are intuitively the most difficult problems among the class NP. NP is the set of problems that can be solved in polynomial-time by nondeterministic Turing machines. Although we do not have a proof yet, many researchers believe that no polynomial-time algorithms exist for any NP-complete problems.[5] Cook's study created a new field of research through which countlessly many combinatorial problems have been found to be NP-complete [10].

By definition, it is trivial that every problem in NP can be solved in exponential time (by a Turing machine). The theory of NP-completeness explicitly and firmly fixed the idea that "polynomial-time algorithms are efficient" in the minds of researchers. We would like to call this idea the *polynomial computation paradigm*.

[5] This is the "P vs. NP problem," which is one of the seven Millennium Prize open problems in mathematics at the end of the twntienth century.

Many important polynomial-time algorithms are now known, including the two basic polynomial-time algorithms for the linear programming problem (LP), namely the ellipsoid method proposed by Khachiyan in 1979 [15] and the interior-point method proposed by Karmarkar in 1984 [13], the strongly polynomial-time algorithm for the minimum cost flow problem proposed by Éva Tardos in 1985 [19], the linear-time shortest path algorithm with positive integer edge lengths proposed by Mikkel Thorup in 1997 [20], and the deterministic polynomial-time algorithm for primality test proposed by Agrawal, Kayal, and Saxena in 2002 [1]. These algorithms pioneered new perspectives in the field of algorithm research. They are gems that were found under the polynomial computation paradigm.

1.3.2 Emergence of Sublinear-Time Algorithms

Although linear-time algorithms have naturally considered the fastest, since intuitively we basically have to read all the data when solving a problem, the new idea of "sublinear-time algorithms" emerged at the end of the twentieth century. Sublinear-time algorithms run by reading only a sublinear (i.e., $o(n)$) amount of data from the input.

The most popular framework for sublinear-time algorithms is "property testing." This idea was first presented by Rubinfeld and Sudan [18] in 1996 (although it appeared even earlier at a conference version in 1992) in the context of program checking. In this paper, they introduced the ideas of "distance" between an instance (e.g., a function) and a property (e.g., linearity), and "ϵ-farness." They also gave constant-time testers for some properties of functions. The first study giving the notion of constant-time testability of combinatorial (mainly graph) structures was given by Goldreich, Goldwasser, and Ron [11], which was present a conference in 1995 (STOC'95). After the turn of the century, many studies that follow this idea of testability have appeared and the importance of this field is growing [2, 9].

1.3.3 Property Testing and Parameter Testing

We say that a *testing algorithm* (or *tester* for short) for a property $\mathcal{P}$ accepts a given instance I with probability at least $2/3$ if I has $\mathcal{P}$ and rejects it with probability at least $2/3$ if I is far from having $\mathcal{P}$. $\mathcal{P}$ is defined as a (generally infinitely large) subset of instances. The distance between I and $\mathcal{P}$ is defined as the minimum Hamming distance between I and $I' \in \mathcal{P}$. The distance is normalized to be in $[0, 1]$ (i.e., $\mathrm{dist}(I, \mathcal{P}) \in [0, 1]$). If an instance has the property, the distance is zero (i.e., $\mathrm{dist}(I, \mathcal{P}) = 0$ if $I \in \mathcal{P}$). If $\mathrm{dist}(I, \mathcal{P}) \geq \epsilon$ for an $\epsilon \in [0, 1]$, then we say that I is ϵ-far from $\mathcal{P}$ and otherwise ϵ-close. A tester rejects I with probability at least $2/3$ if I is ϵ-far from $\mathcal{P}$.

For a property, if a tester exists whose running time[6] is bounded by a constant independent of the size of the input, then we call the property is *testable*.[7] This framework is called *property testing*.

Property testing is a relaxation of the framework of decision problems. In contrast, a relaxation of the framework of optimization problems is *parameter testing*. In parameter testing, we try to find an approximation to the value of the objective function with an additive error of at most ϵN from the optimum value, where N is the maximum value of the objective function.

This idea appeared at the end of the twentieth century, and was further developed in this century. See Chaps. 2 and 3 for these themes.

1.4 Ways to Decrease Computational Resources

In addition to property and parameter testing, there are various methods for decreasing the amount of computational resources needed for handling big data. Although some methods may require linear computation, each of them has strong merits. We briefly introduce these methods in this section.

1.4.1 Streaming Algorithms

Property testing generally uses the assumption that an algorithm can read any position (cell) of the input. However, this may be difficult in some situations, such as if the data arrives as a stream (sequence) and the algorithm is required to read the values one by one in the order of arrival. The key assumption of this framework is that an algorithm does not have enough memory to store the entire input. For example, to find the maximum value in a sequence of integers $a_1, \ldots, a_n$, it is enough to use $O(1)$ cells of memories.[8]

Although this method requires linear computation time, since it must read all of the data, the amount of memory is constant in many cases. If we assume that the order of data arrival in the stream is random, then it becomes close to the setting of (nonadaptive[9]) property testing. In this book, streaming algorithms are covered in Chap. 16.

---

[6] Normally we also use the "query complexity" besides the running time. See Chap. 2 for details.

[7] Sometimes "testable" means that the problem has an algorithm with a sublinear query complexity, and *strongly testable* may be used for distinguishing constant query complexity from mere sublinear query complexity.

[8] We assume that each memory cell can store any one integer among $\{a_1, \ldots, a_n\}$.

[9] *Nonadaptive* means that the query (of an algorithm) cannot depend on any answer of the queries; in other words, the queries are fixed before the algorithm starts.

1.4.2 Compression

Compression is a traditional and typical method for treating digital data. Basically, there are two types of compression: one type is compression of data without losing any information. In this type of compression, there is an information-theoretical lower bound on the data size. This method is used when the original data needs be reconstructed perfectly from the compressed data, and it thus called *lossless compression* or *reversible compression*. The other type of compression allows discarding of some of the data such that the compressed data is an inexact approximation. Although some of these algorithms can compress data drastically, it is not possible to reconstruct the original data perfectly from the compressed data, and these algorithms are thus called *lossy compression* or *irreversible compression*. This method works remarkably well in the field of music and image compression. See Chaps. 6, 7, 10, and 16 in this book for results from this area.

1.4.3 Succinct Data Structures

When compressed data is used, it essentially needs to be decompressed. However, decompression requires extra computation. It is therefore useful to be able to use compressed data as-is without decompression. Succinct data structures are a framework that realizes this idea. Specifically, succinct data structures use an amount of space that is close to the information-theoretical lower bound while still allowing efficient (fast) query operations. These structures involve a tradeoff between space and time. See Chaps. 8 and 9 for details.

1.5 Need for the Sublinear Computation Paradigm

1.5.1 Sublinear and Polynomial Computation Are Both Important

Even though the sublinear computation paradigm has become necessary, it does *not* mean that the polynomial computation paradigm is obsolete. Polynomial computation is still important in normal computations. The typical cases where the sublinear computations are needed are when we need to treat big data. In such cases, traditional polynomial computation is sometimes too slow.

This relationship between the polynomial computation paradigm and the sublinear computation paradigm is analogous to the relationship between *Newtonian mechanics* and *the theory of relativity* in physics. While Newton mechanics is used for normal physical calculations, the theory of relativity is needed if we try to calculate the motion of very fast objects such as rockets, satellites, or electrons. We entered

the era of the theory of relativity in the twentieth century and the era of sublinear computation era in the twenty-first century.

1.5.2 Research Project ABD

A research project named "Foundations on Innovative Algorithms for Big Data (ABD),"[10] in which the sublinear computation paradigm is the central concept was started by JST, CREST, Japan in October 2014 and concluded in September 2021. The total budget was more than 300 million yen. Although the project had 24 members at its inception, many more researchers later joined and the final number of regular members exceeded 40 in total. The leader of the project was Prof. Naoki Katoh of University of Hyogo.[11] The project consisted of three groups: the Sublinear Algorithm Group (Team A) led by Prof. Katoh; the Sublinear Data Structure Group (Team D) led by Prof. Tetsuo Shibuya of the University of Tokyo; and the Sublinear Modeling Group (Team M) led by Prof. Kazuyuki Tanaka of Tohoku University. In this project, we worked on problems in big data computation. The main purpose of this book is to introduce the results of this project. A special issue of The Review of Socionetwork Strategies [14] is also available for this project. While some of the methods adopted in this project are not sublinear, we are confident that every piece of research concluded under the project is useful and will form the foundations of innovative algorithms for big data!

1.5.3 The Organization of This Book

This part of the book, Part I, has provided an introduction. Parts II, III, and IV present the theoretical results of Teams A, D, and M, respectively. Application results leading to scientific and technological innovation are compiled in Part V.

References

1. M. Agrawal, N. Kayal, N. Saxena, PRIMES P. Ann. Math. **160** (2004)
2. A. Bhattacharyya, Y. Yoshida, Property Testing—Problems and Techniques (Springer, 2021) (to be published in 2021)
3. A. Cobham, The intrinsic computational difficulty of functions, Logic, methodology and philosophy of science, in *Proceedings of the 1964 International Congress*, ed. by Y. Bar-Hillel,

[10] http://crest-sublinear.jp/en/.

[11] Although he was at Kyoto University when the project started, he later moved to Kwansei Gakuin University because of his retirement before finally moving again to his present position.

Studies in Logic and the Foundations of Mathematics (North-Holland Publishing Company, Amsterdam, 1965), pp. 24–30
4. Cobham's thesis, Wikipedia, https://en.wikipedia.org/wiki/Cobham%27s_thesis#cite_note-7
5. S. A. Cook,The complexity of theorem proving procedures, in *Proceedings of the STOC'71* (1971), pp. 151–158
6. E.W. Dijkstra, A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
7. J. Edmonds, Jack, Paths, trees, and flowers. Can. J. Math. **17**, 449–467 (1965)
8. L.R. Ford, D.R. Fulkerson, Maximal flow through a network. Can. J. Math. **8**, 399–404 (1956)
9. O. Goldreich, *Introduction to Property Testing* (Cambridge University Press, 2017)
10. M.R. Garey, D.S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness* (W. H, Freeman and Company, 1979)
11. O. Goldreich, S. Goldwasser, D. Ron, Property testing and its connection to learning and approximation. J. ACM **45**(4), 653–750 (1998)
12. O. Goldreich, D. Ron, Property testing in bounded degree graphs. Proc. STOC **1997**, 406–415 (1997)
13. N. Karmarkar, A new polynomial time algorithm for linear programming. Combinatorica **4**, 373–395 (1984)
14. N. Katoh et. al. (eds.), Special issue on foundations of innovative algorithms for big data—sublinear computational paradigm and its expansions. Rev. Socionetwork Strategies **13**(2), (2019)
15. L.G. Khachiyan, A polynomial algorithm in linear programming. Doklady Akademii Nauk SSSR **244**, 1093–1096 (1979)
16. J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. AMS **7**(1), 48–50 (1956)
17. R.C. Prim, Shortest connection networks and some generalizations. Bell Syst. Tech. J. **36**, 1389–1401 (1957)
18. R. Rubinfeld, M. Sudan, Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. Vo. **25**(2), 252–271 (1996)
19. É. Tardos, A strongly polynomial minimum cost circulation algorithm. Combinatorica **3**, 247–255 (1985)
20. M. Thorup, Undirected single source shortest paths with positive integer weights in linear time. J. ACM **46**, 362–394 (1999) (Conference version was in FOCS'97)
21. "Total number of Websites," internet lives starts. https://www.internetlivestats.com/total-number-of-websites/
22. "We knew the web was big...," Google Official Blog. https://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html