IEICE TRANS. COMMUN., VOL.E95–B, NO.5 MAY 2012

PAPER

Performance of Thorup's Shortest Path Algorithm for Large-Scale Network Simulation

Yusuke SAKUMOTO[†a)], **Hiroyuki OHSAKI**[††b)], *and* **Makoto IMASE**[††c)], *Members*

SUMMARY In this paper, we investigate the performance of Thorup's algorithm by comparing it to Dijkstra's algorithm for large-scale network simulations. One of the challenges toward the realization of large-scale network simulations is the efficient execution to find shortest paths in a graph with N vertices and M edges. The time complexity for solving a single-source shortest path (SSSP) problem with Dijkstra's algorithm with a binary heap (DIJKSTRA-BH) is $O((M+N)\log N)$. An sophisticated algorithm called *Thorup's algorithm* has been proposed. The original version of Thorup's algorithm (THORUP-FR) has the time complexity of $O(M + N)$. A simplified version of Thorup's algorithm (THORUP-KL) has the time complexity of $O(M\alpha(N) + N)$ where $\alpha(N)$ is the functional inverse of the Ackerman function. In this paper, we compare the performances (i.e., execution time and memory consumption) of THORUP-KL and DIJKSTRA-BH since it is known that THORUP-FR is at least ten times slower than Dijkstra's algorithm with a Fibonaccii heap. We find that (1) THORUP-KL is almost always faster than DIJKSTRA-BH for large-scale network simulations, and (2) the performances of THORUP-KL and DIJKSTRA-BH deviate from their time complexities due to the presence of the memory cache in the microprocessor.

key words: *SSSP (single-source shortest path problem), large-scale network simulation, Dijkstra's algorithm, Thorup's algorithm*

1. Introduction

One of the challenges toward the realization of large-scale network simulations is the efficient execution of shortest-path routing (e.g., static routing) [1]–[3]. All network simulators require some type of routing to be performed. Network simulators generally use a solution that assumes a single-source shortest path (SSSP) problem with static routing. For instance, one of the most popular network simulators, ns-2 [4], uses Dijkstra's algorithm with a binary heap (hereafter referred to as DIJKSTRA-BH) [5], [6]. With static routing, ns-2 first obtains the shortest paths for all source–destination node pairs.

Note the mapping between static routing in network simulation and an SSSP problem in graph theory. A *network* consisting of *nodes* and *links* can, in network simulation, be viewed as a *graph* consisting of *vertices* and *edges* permitting the application of graph theory. In this paper, we intentionally use both types of words according to the context.

Manuscript received September 20, 2011.

[†]The author is with the Faculty of System Design Tokyo Metropolitan University, Hino-shi, 191-0016 Japan.

[††]The authors are with the Graduate School of Information Science and Technology Osaka University, Suita-shi, 565-0871 Japan.

a) E-mail: sakumoto@sd.tmu.ac.jp

b) E-mail: oosaki@ist.osaka-u.ac.jp

c) E-mail: imase@ist.osaka-u.ac.jp

DOI: 10.1587/transcom.E95.B.1592

The time complexity of DIJKSTRA-BH is $O((M + N)\log N)$ where N and M are the number of vertices and edges, respectively, in a graph [5], [6]. Since network simulators usually need to obtain the shortest paths for all source–destination node pairs, network simulators take a significant amount of time just for static routing [1].

An sophisticated solution for the single-source shortest path problem, called *Thorup's algorithm*, has been proposed [7]. The original version of Thorup's algorithm (hereafter THORUP-FR) uses Fredman's algorithm [8] for obtaining the minimum spanning tree. THORUP-FR is a linear time algorithm with the time complexity of $O(M + N)$. A simplified version of Thorup's algorithm (hereafter THORUP-KL), which uses Kruskal's algorithm [9] for obtaining the minimum spanning tree, is also discussed in [7]. The time complexity of THORUP-KL is $O(M\,\alpha(N) + N)$ where $\alpha(N)$ is the functional inverse of the Ackerman function [6]. The time complexity of THORUP-KL is larger than that of THORUP-FH.

A comparison of the time complexities of Thorup's and Dijkstra's algorithms suggests that Thorup's algorithm is more efficient than Dijkstra's for, in particular, large-scale network simulations. However, to the best of our knowledge, this efficiency in practice has not been fully investigated for large-scale network simulations.

The objective of this paper is, therefore, to investigate the performance of Thorup's algorithm by comparing it to Dijkstra's, and to answer the following questions.

1. How efficiently/inefficiently does Thorup's algorithm perform compared with Dijkstra's for large-scale (e.g., million nodes) network simulations?
2. How and why does the practical performance of Thorup's and Dijkstra's algorithms deviate from their time complexities (i.e., theoretical performance)?

The work by Asano et al. [10] reported that practical efficiency of THORUP-FR is at least ten times lower than that of Dijkstra's algorithm with a Fibonacci heap (hereafter DIJKSTRA-FH) [11]. DIJKSTRA-FH has a better amortized time complexity than DIJKSTRA-BH [5]. In [10], the authors investigated the performance of THORUP-FR (i.e., the *original* version of Thorup's algorithm) using their implementation for medium-scale random graphs with 50,000 vertices. Even with the linear time complexity of THORUP-FR, the authors showed that THORUP-FR is at least ten times slower than DIJKSTRA-FH. The authors explain that their implementation of THORUP-FR is very slow due to

Copyright © 2012 The Institute of Electronics, Information and Communication Engineers

the time needed to construct the minimum spanning tree with Fredman's algorithm [10].

In this paper, we compare the performances (i.e., execution time and memory consumption) of THORUP-KL (i.e., a *simplified* version of Thorup's algorithm) with the time complexity of $O(M\,\alpha(N) + N)$ and DIJKSTRA-BH.

There are several variants of Dijkstra's algorithm with different time complexities of $O(M + N^2)$ [12], $O((M + N)\log N)$ [5] and $O(M+N\log N)$ [11]. In this paper, we focus on Dijkstra's algorithm with a binary heap (DIJKSTRA-BH) [5] with the time complexity of $O((M+N)\log N)$. Note that the communication network considered here is a sparse network. Although DIJKSTRA-FH has the smallest order of time complexity $O(M+N\log N)$ among the variants known, it is known that DIJKSTRA-BH is faster than DIJKSTRA-FH for sparse networks [13], [14]. Therefore, DIJKSTRA-BH is the best choice to provide a comparison of THORUP-KL.

Extensive experiments on our implementations of THORUP-KL and DIJKSTRA-BH elucidate their execution time and memory consumption. Among our findings (1) THORUP-KL is almost always faster than DIJKSTRA-BH for large-scale network simulations, and (2) the performances of THORUP-KL and DIJKSTRA-BH deviate from their time complexities due to the presence of memory cache in the microprocessor.

The organization of this paper is as follows. Section 2 introduces Thorup's algorithm. In Sect. 3, we compare the performance of DIJKSTRA-BH and THORUP-KL for large-scale network simulations. Finally, Sect. 4 concludes this paper and discusses future works.

2. Thorup's Algorithm

In this section, we briefly introduce Thorup's algorithm. Refer to [7] for the details. Thorup's algorithm is a solution of the single-source shortest path (SSSP) problem for an undirected graph $G = (V, E)$ with source vertex $s \in V$ and positive integer edge weight function $l : E \to \{1, \cdots, 2^\omega, \infty\}$ where ω is the word length [7]. If $(v, w) \notin E$, we define $l(v, w) = \infty$.

The single source shortest path problem (SSSP) is to find the shortest paths with the distance $d(v)$ from source vertex s to every vertex $v \in V$. Let D be the super distance, which follows that $D(v) = d(v)$ for all $v \in S$ and $D(v) = \min_{u\in S} d(u) + l(u, v)$ for all $v \in V - S$. If $V = S$ (i.e., $d(v) = D(v)$ for all vertices), the single source shortest path problem is solved.

Dijkstra's algorithm finds the shortest paths from a source vertex s by gradually expanding S. Initially, $S = \{s\}$, $D(s) = d(s) = 0$, and $D(v) = d(s) + l(s, v)$ for all $v \in V - \{s\}$. Dijkstra proved in [12] that

$$D(v) = d(v), \tag{1}$$

if vertex $v \in V-S$ minimizes $D(v)$. Hence, since $D(v) = d(v)$ for the vertex v minimizing $D(v)$, the vertex v can be moved to S. At the same time, Dijkstra's algorithm visits the vertex v, and if $D(v) + l(v, w) < D(w)$ for any $w \in V - S$, Dijkstra's algorithm decreases $D(w)$ to $D(v) + l(v, w)$. Dijkstra's algorithm repeats visiting vertex $v \in V - S$ minimizing $D(v)$ until $S = V$. For visiting vertex $v \in V - S$ minimizing $D(v)$, Dijkstra's algorithm needs to sort vertices according to $D(v)$, leading to non-linear time complexity.

Thorup's algorithm uses *component hierarchy* of G for realizing linear time complexity [7]. We denote the subgraph of G by $G_i = (V, E_i)$ where $E_i = \{(u, w) \in E | l(u, w) < 2^i\} (0 \le i \le \omega)$. Hence, G_0 does not have a edge. G_ω is equal to G. On level i in the component hierarchy, we have the components (maximal connected subgraphs) of G_i. We denote the component on level i by $[v]_i$ including the vertex v. If $w \in \mathrm{Ver}([v]_i)$, $[v]_i = [w]_i$. Let $\mathrm{Ver}([v]_i)$ be the set of vertices of the component $[v]_i$. Let $\min([v]_i)$ be $\min_{w\in \mathrm{Ver}([v]_i)-S} D(w)$ for given G and S. If $\mathrm{Ver}([v]_i) - S = \emptyset$, $\min([v]_i) = \infty$. We will write $x \gg i$ is right shift operation, which calculates the i least significant bits of x out to the right. Thorup proved in [7] that

$$D(v) = d(v), \tag{2}$$

if vertex $v \in V-S$ satisfies $\min([v]_i) \gg i-1 = \min([v]_{i-1}) \gg i - 1$ for any $i (0 < i \le \omega)$. Hence, in Thorup's algorithm, regardless of whether vertex v does not minimizes $D(v)$, the vertex v can be moved to S. Therefore, Thorup's algorithm does not need to the sorting vertices according to $D(v)$. Component $[v]_i$ $(i > 0)$ has a bucket to obtain components with $\min([v]_i) \gg i - 1 = \min([v]_{i-1}) \gg i - 1$. For efficiently obtaining the components, Thorup's algorithm sorts components $[v]_{i-1}$ in the bucket of $[v]_i$ according to $\min([v]_{i-1}) \gg i - 1$ with bucket sort.

In Thorup's algorithm, the topological structure of the component hierarchy represents *component tree*. Each node on the level i in a component tree corresponds to a component on the level i in a component hierarchy. $[v]_\omega$ is the root node in a component tree. If $\mathrm{Ver}([v]_i) \subset \mathrm{Ver}([v]_{i+1})$ in a component hierarchy, $[v]_{i+1}$ is the parent of $[v]_i$ in a component tree. If $\mathrm{Ver}([v]_i) = \mathrm{Ver}([v]_{i-1})$, $[v]_{i-1}$ is skipped in a component tree. Figures 1 and 2 illustrate examples of the graph and component tree, respectively. When the source vertex $s = v_1$, initially, $S = \{v_1\}$, $D(v_1) = d(v_1) = 0$, $D(v_2) = 5$, $D(v_4) = 4$, and $D(v_3) = D(v_5) = \infty$. In Thorup's algorithm, $\min([v_1]_3) = \min([v_2]_2) = \min([v_4]_1) = 4$, $\min([v_2]_1) = \min([v_2]_0) = 5$, and $\min(\cdot) = \infty$ for otherwise components. Since the vertex v_4 minimizes $D(v)$, Dijkstra's algorithm can visit the vertex v_4. On the other hand, since $\min([v_2]_3) \gg 2 = \min([v_2]_2) \gg 2$, $\min([v_2]_2) \gg 1 = \min([v_2]_1) \gg 1$, $\min([v_2]_1) \gg 0 = \min([v_2]_0) \gg 0$, $\min([v_4]_3) \gg 2 = \min([v_4]_2) \gg 2$, and $\min([v_4]_2) \gg 1 = \min([v_4]_1) \gg 1$, Thorup's algorithm can visit the vertices v_2 and v_4.

Figure 3 illustrates an example of building a component tree in Thorup's algorithm. First, the component $[v_1]_3$ for the whole graph is added to the component tree (see Fig. 3(b)). The graph is partitioned into three components, $[v_1]_2$, $[v_2]_2$, and $[v_5]_2$ by ignoring edges $(u, w) \in E - E_2$ (see Fig. 3(c)). The components $[v_1]_2$, $[v_2]_2$, and $[v_5]_2$ are added

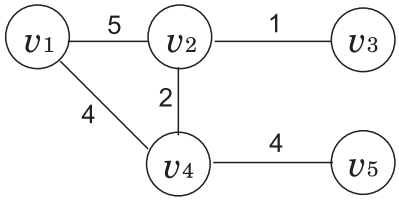

Fig. 1 An example of a graph.

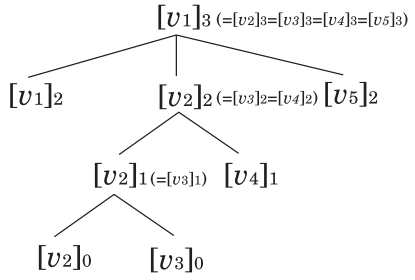

Fig. 2 An example of the component tree for the graph illustrated in Fig. 1.

to the component tree (see Fig. 3(d)). This procedure is repeated until all components become singletons. Namely, edges $(u, w) \in E - E_0$ are then ignored, which results in five singletons, $[v_1]_0$, $[v_2]_0$, $[v_3]_0$, $[v_4]_0$, and $[v_5]_0$ (see Fig. 3(g)). The components $[v_1]_0$ through $[v_5]_0$ are added to the component tree (see Fig. 3(h)). In Fig. 2, the components $[v]_{i-1}$ with $[v]_i = [v]_{i-1}$ are skipped.

Figure 4 illustrates an example of finding shortest paths from the vertex v_1 (source vertex) to all other vertices in Thorup's algorithm. Initially, $S = \{v_1\}$, $D(v_1) = d(v_1) = 0$, $D(v_2) = 5$, $D(v_4) = 4$, $D(v_3) = D(v_5) = \infty$, $\min([v_1]_3) = \min([v_2]_2) = \min([v_4]_1) = 4$, $\min([v_2]_1) = \min([v_2]_0) = 5$, and $\min(\cdot) = \infty$ for otherwise components (see Fig. 4(a)). First, since $\min([v_2]_3) \gg 2 = \min([v_2]_2) \gg 2$, $\min([v_2]_2) \gg 1 = \min([v_2]_1) \gg 1$, $\min([v_2]_1) \gg 0 = \min([v_2]_0) \gg 0$, $\min([v_4]_3) \gg 2 = \min([v_4]_2) \gg 2$, and $\min([v_4]_2) \gg 1 = \min([v_4]_1) \gg 1$, Thorup's algorithm can visit the vertices v_2 and v_4 (see Fig. 4(b)). In this example, Thorup's algorithm visits the vertex v_2 in first. Secondly, Thorup's algorithm visits the vertex v_4(see Fig. 4(d)). Thirdly, since $\min([v_3]_3) \gg 2 = \min([v_3]_2) \gg 2$, $\min([v_3]_2) \gg 1 = \min([v_3]_1) \gg 1$, and $\min([v_3]_1) \gg 0 = \min([v_3]_0) \gg 0$, Thorup's algorithm can visit the vertex v_3 (see Fig. 4(f)). Finally, Thorup's algorithm visits the vertex v_5 (see Fig. 4(h)).

For archiving the time complexity $O(M + N)$ of finding shortest paths, Thorup's algorithm determines bucket sizes of each component by using the minimum spanning tree of G. Thorup proved that the total amount of bucket sizes is $O(M + N)$ in [7].

The component tree is a N-ary tree [15]. A component $[v]_i$ on the level i of the component tree has pointers for child component $[w]_{i-1}$ on the level $i - 1$, and a bucket to store its pointers according to $\min([w]_{i-1})$ of child component $[w]_{i-1}$. Since each component has a bucket, the component tree consumes more memory compared with binary heap in DIJKSTRA-BH. To easily implement traversing the component tree from the component tree, recursion is needed.

The efficiency of Thorup's algorithm is due to minimum spanning tree. In [7], two algorithms for finding the minimum spanning tree, Kruskal's [9] and Fredman's [8] algorithms, are discussed.

Fredman's algorithm is an efficient algorithm, whose time complexity is $O(M)$. The original version of Thorup's algorithm (THORUP-FR) uses Fredman's algorithm for obtaining the minimum spanning tree. Incorporation of Fredman's algorithm into THORUP-FR is another key technique for realizing a linear time algorithm of $O(M + N)$.

Kruskal's algorithm is an simple algorithm with the time complexity of $O(M\,\alpha(N))$ where $\alpha(N)$ is the functional inverse of the Ackerman function [6]. A simplified version of Thorup's algorithm (THORUP-KL) uses Kruskal's algorithm for obtaining the minimum spanning tree. THORUP-KL is no longer a linear time algorithm since Kruskal's algorithm is not a linear time algorithm; i.e., the time complexity of THORUP-KL is $O(M\,\alpha(N) + N)$.

3. Experiment

3.1 Methodology

Through extensive experiments, we compare the performance of Thorup's and Dijkstra's algorithms in terms of speed and memory consumption. We implemented Thorup's algorithm using Kruskal's algorithm for obtaining the minimum spanning tree (THORUP-KL) and Dijkstra's algorithm using with binary heap (DIJKSTRA-BH) [6].

Note that in our experiments, THORUP-KL is used instead of THORUP-FR even though THORUP-KL is theoretically less efficient than THORUP-FR. As explained in Sect. 2, THORUP-KL is not a linear time algorithm. However, we intentionally use THORUP-KL instead of THORUP-FR for the following two practical reasons.

The first reason is that Asano et al. have shown that THORUP-FR is very slow in practice [10]. The second reason is that the time complexity of Kruskal's algorithm, $O(M\,\alpha(N))$, should be comparable with that of Fredman's, $O(M)$, since it is known that $\alpha(N) \leq 4$ for all $N < 2^{2^{2^{65536}}} - 3$ [6]. Namely, the performance of THORUP-KL should be comparable with (or might be even faster than) that of THORUP-FR for large-scale network simulations.

All experiments were run on a computer with a single Intel Pentium4 2.80 [GHz] processor with 1 [Gbyte] of memory running Debian GNU/Linux 5.0 (kernel version 2.6.26).

For clearly examining performance of THORUP-KL and DIJKSTRA-BH, we used a simple network topology. Namely, we generated a random network with ER (Erdös-Rényl) model [16] for given network size N (i.e., the number of nodes) and average degree k (i.e., the average number of links connected to each node). A random network is a traditional network model, and it has been used in large-scale network simulations [17]. We claim neither that a ran-

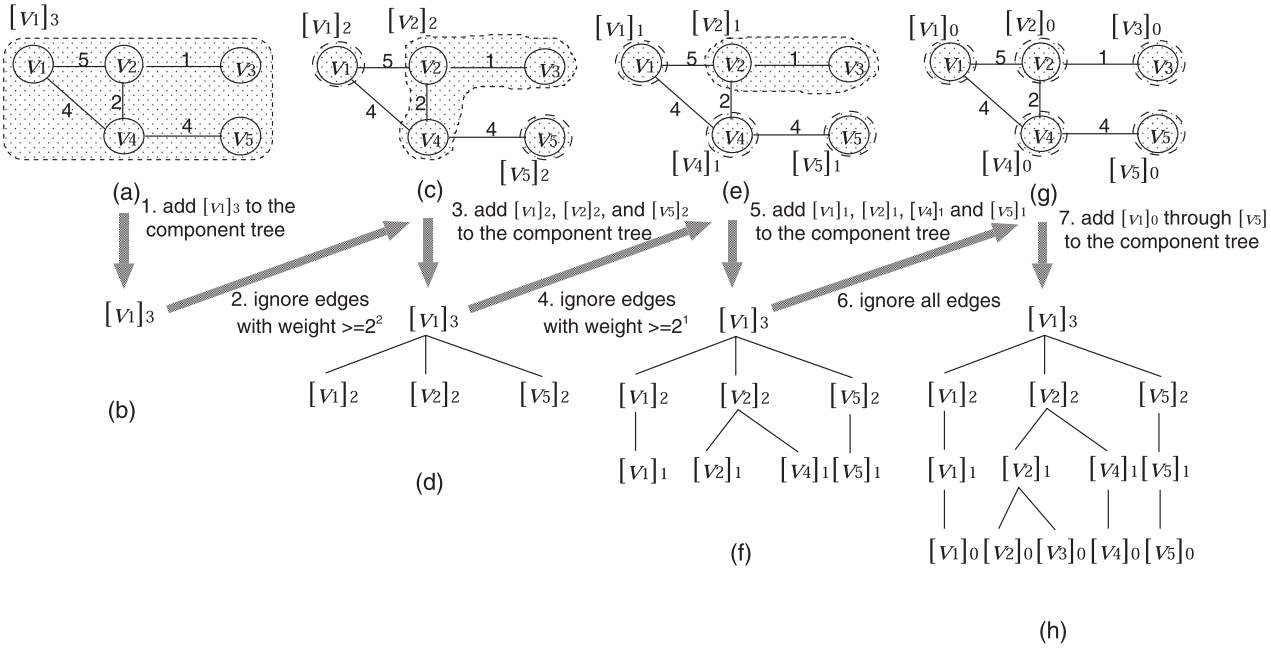

Fig. 3 An example of building the component tree.

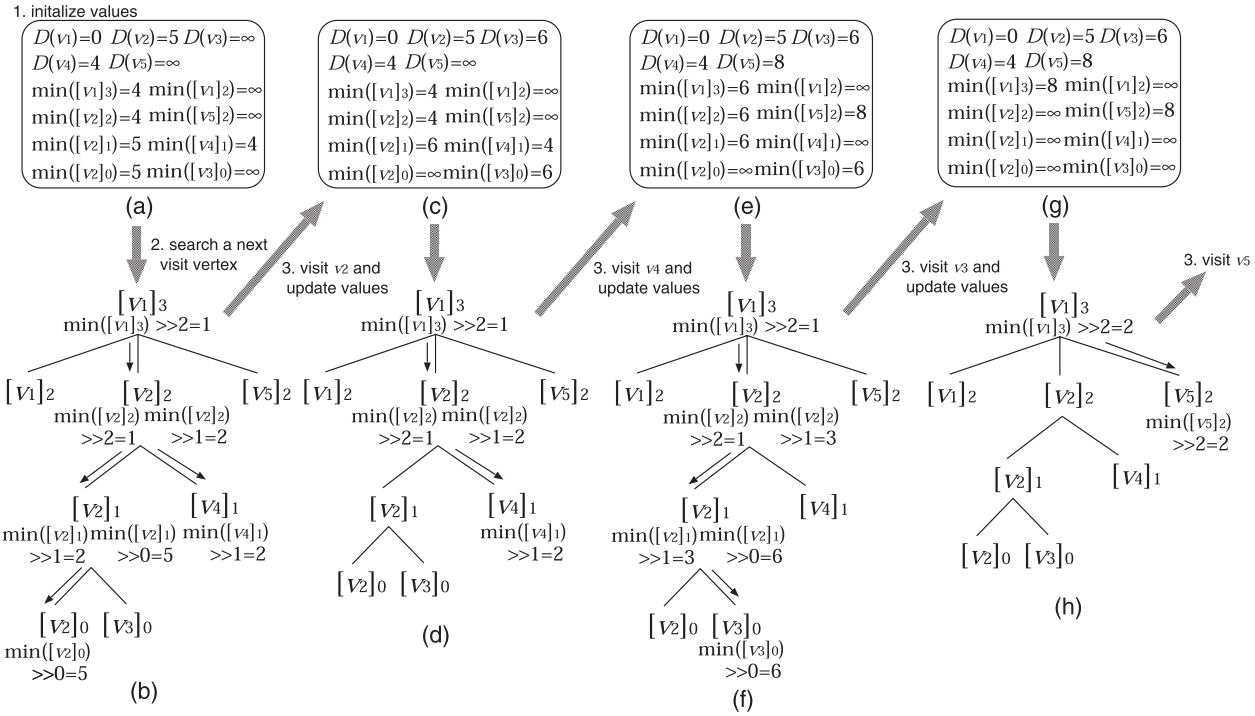

Fig. 4 Finding shortest paths from the vertex v_1 using the component tree.

dom network is the *typical* network topology for network simulation studies nor that performance evaluation with a random network is sufficient for comparing performance of THORUP-KL and DIJKSTRA-BH. However, similar to Asano et al. [10], we generated a random network to simplify the performance evaluation of THORUP-KL.

We measured the average and 95% confidence interval of measurements (e.g., execution time and memory consumption) for ten random graphs with the same network size N and average degree k. In all figures, 95% confidence intervals are not shown since they are negligible small (i.e., less than 1.0% of each measurement).

3.2 Speed

We first evaluate the performance of THORUP-KL and DIJKSTRA-BH in terms of speed by measuring their execution times. THORUP-KL and DIJKSTRA-BH are for a single-source shortest path problem. On the contrary, network simulators usually need to obtain shortest paths for all source–destination node pairs. We, therefore, estimated the execution time for obtaining *all-pairs* shortest paths from the execution time taken to obtain single-source shortest paths. Note that obtaining all-pairs shortest paths is the worst case; i.e., if the number of source–destination pairs is not so large, the network simulator does not need to obtain all-pairs shortest paths. We separately measured the mean times for program initialization, $\overline{T^G_{init}}$, and the computation of single-source shortest paths, $\overline{T^G_{SSSP}}$, for graph G. The execution time for obtaining all-pairs shortest paths for G with N vertices, T^G_{APSP}, is estimated as

$$T^G_{APSP} = \overline{T^G_{init}} + N\,\overline{T^G_{SSSP}}.$$

Execution times of THORUP-KL and DIJKSTRA-BH for obtaining all-pairs shortest paths for different network sizes with an average degree (i.e., $k = 5$) and multiple edge weights (i.e., randomly-chosen integer edge weights from 1 to 1,000) are shown in Fig. 5. Note that the average degree of Internet AS topology was 5.97 in 2005 [18], [19]. This figure shows that execution times increase rapidly as network size N increases. This phenomenon is not surprising since the time complexity of any algorithm for all-pairs shortest path problem is at least $O(N^2)$.

To visually show the difference in execution times of THORUP-KL and DIJKSTRA-BH, the relative execution time (i.e., the execution time of THORUP-KL normalized by that of DIJKSTRA-BH) for $k = 5$ and 10 is shown in Fig. 6. In this figure, results with two types of edge weights are shown: the single edge weight (i.e., 1 for all edges) and multiple edge weights (i.e., randomly-chosen integer edge weights from 1 to 1,000).

In contrast to the comparison results of THORUP-FR and DIJKSTRA-FH in [10], this figure clearly indicates that THORUP-KL is almost always faster than DIJKSTRA-BH for any network size N, average degree k, and edge weight. In [10], it was reported that even with its linear time complexity, THORUP-FR is at least ten times slower than DIJKSTRA-FH for medium-scale random graphs with 50,000 vertices. In Fig. 6, the maximum value of the relative execution time is 1.02, which implies that the performance of THORUP-KL is comparable with that of DIJKSTRA-BH even in the worst case. For large-scale (e.g., million nodes) network simulations, THORUP-KL achieves almost double the efficiency of DIJKSTRA-BH.

These results suggest a natural question: why does THORUP-FR perform inefficiently in [10] while THORUP-KL performs efficiently in our experiments? One possible explanation is the algorithm used for finding the minimum spanning tree. Namely, Asano et al. used Fredman's algorithm whereas we used Kruskal's algorithm. It was reported in [10] that for random graphs with 50,000 vertices, Fredman's algorithm itself consumes significant amount of time (e.g., 86% for $M = 175,065$ and 98% for $M = 424,396$) of the total execution time in their experiments.

From detailed measurements of our implementation, we have found that Kruskal's algorithm itself consumed approximately 5% of the total execution time. Such a drastic reduction in the execution time for finding the minimum spanning tree should be the reason for effectiveness of THORUP-KL. Note that Asano et al. addressed and extensively studied the practical inefficiency of Fredman's algorithm in [10]. We could choose THORUP-KL with Kruskal's algorithm due to their findings.

Also, contrary to one's expectation, Fig. 6 shows a somewhat strange phenomenon; i.e., the relative execution time is like a *camel-shaped function* for network size N regardless of the average degree k and the type of edge weights.

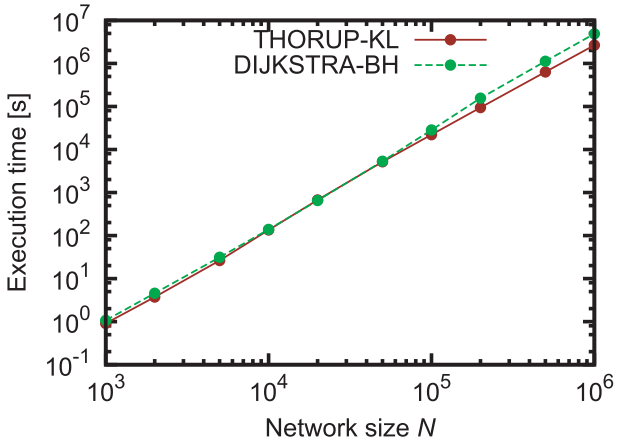

Fig. 5 Execution times of THORUP-KL and DIJKSTRA-BH for obtaining all-pairs shortest paths for the average degree $k = 5$ and multiple edge weights of 1–1000.

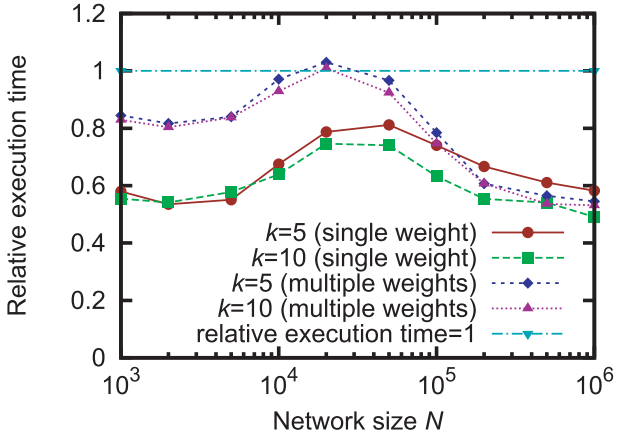

Fig. 6 Relative execution time (i.e., the execution time of THORUP-KL normalized by that of DIJKSTRA-BH) for different average degrees k and types of edge weights.

The relative time complexity of THORUP-KL to DIJKSTRA-BH is given by

$$\frac{O((M\,\alpha(N)+N)}{O((M+N)\,\log N)} \approx O\left(\frac{\alpha(N)}{\log N}\right). \quad (3)$$

Note the existence of $\alpha(N)$ in the numerator because we used Kruskal's algorithm instead of Fredman's. Recall that $\alpha(N) \le 4$ for all $N < 2^{2^{2^{65536}}} - 3$ [6]. Thus, Eq. (3) should be an asymptotically monotone decreasing for network size N. One would, therefore, expect that the relative execution time T^G_{APSP} should yield an asymptotically monotone decreasing function for network size N.

Aside from the camel-shaped function, Fig. 6 indicates that the relative execution time is moderately affected by the type of edge weights, and slightly by the average degree k. This is again contrary to one's expectations. The relative time complexity (Eq. (3)) is independent of the average degree k (i.e., the number of edges M). One would, therefore, expect that relative execution time T^G_{APSP} should not be affected by the average degree k.

In Sects. 3.4 and 3.5, we will investigate the cause of those deviations of the relative execution time (i.e., camel-shaped function and dependence on the average degree k) from the relative time complexity.

3.3 Memory Consumption

We next evaluate the performance of THORUP-KL and DIJKSTRA-BH in terms of memory consumption. We measured the memory consumption for obtaining all-pairs shortest paths. Note that the memory consumption for obtaining all-pairs shortest paths is equivalent to that for obtaining single-source shortest paths because of its repetitive program execution.

Memory consumptions of THORUP-KL and DIJKSTRA-BH for obtaining all-pairs shortest paths for different network sizes are shown in Fig. 7. In this figure, the average degree $k = 5$ and multiple edge weights of 1–1000 are used. Note that we have observed that the memory consumptions of THORUP-KL and DIJKSTRA-BH are not significantly affected by the average degree k and the type of edge weights.

Figure 7 shows that the memory consumption of THORUP-KL is approximately 1.4 times larger than that of DIJKSTRA-BH. This is directly caused by the difference in the data structures used in THORUP-KL (i.e., a hierarchical bucketing structure) and DIJKSTRA-BH (i.e., a binary heap). More specifically, the space complexity [20] of DIJKSTRA-BH is $O(N)$ [5]. The space complexity of THORUP-KL is also $O(N)$ [7]. Therefore, the relative memory consumption (i.e., the memory consumption of THORUP-KL normalized by that of DIJKSTRA-BH) should be independent of network size N. Although the data structure of DIJKSTRA-BH (i.e., a binary heap) can be implemented as one-dimensional array, the data structure of THORUP-KL is a complicated hierarchical bucketing structure. A hierarchical bucketing structure consists of a component tree and buckets for each component, which is apparently less memory efficient than the binary heap.

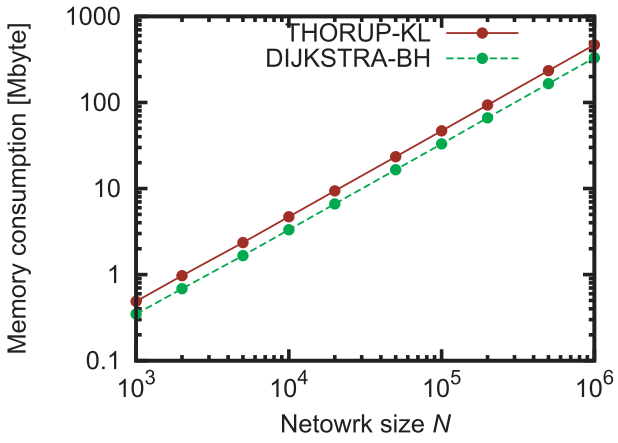

Fig. 7 Memory consumptions of THORUP-KL and DIJKSTRA-BH for the average degree $k = 5$ and multiple edge weights of 1–1000.

3.4 Cause of Camel-Shaped Function

The larger memory consumption of THORUP-KL than that of DIJKSTRA-BH might be the cause of the camel-shaped function of the relative execution time (see Fig. 6). As observed in Fig. 7, THORUP-KL consumes approximately 1.4 times more memory than that of DIJKSTRA-BH. One possible explanation is the effect of memory cache of the microprocessor. If the memory usage is greedy, memory accesses are less likely to hit the cache, significantly slow down program execution.

To clarify why the relative execution time exhibits a camel-shaped function for network size N, we investigate the effect of the memory cache on the performance of THORUP-KL and DIJKSTRA-BH. We performed the same experiments as those in Sect. 3.2 with the memory cache in the microprocessor is disabled. Relative execution times with and without the memory cache are shown in Fig. 8.

Figure 8 clearly shows that the relative execution time without the memory cache is a monotone decreasing function for network size N. On the contrary, when the memory cache is enabled, the relative execution time is a camel-shaped function for network size N.

Figure 8 also plots a fitted curve for the relative execution time without the memory cache. Recall again the relative time complexity (Eq. (3)) of THORUP-KL to DIJKSTRA-BH and $\alpha(N) \le 4$ for all $N < 2^{2^{2^{65536}}} - 3$. We therefore used $c_1/\log N + c_2$ for curve fitting, and obtained $c_1 = 7.17$ and $c_2 = 0.08$. The fitted curve well matches the relative execution time without the memory cache, indicating that the relative execution time can be well explained with the relative time complexity (Eq. (3)). Reversely, this suggests that the camel-shaped function of the relative execution time is due to be the memory cache of the microprocessor.

More detailed investigation can be possible by examin-

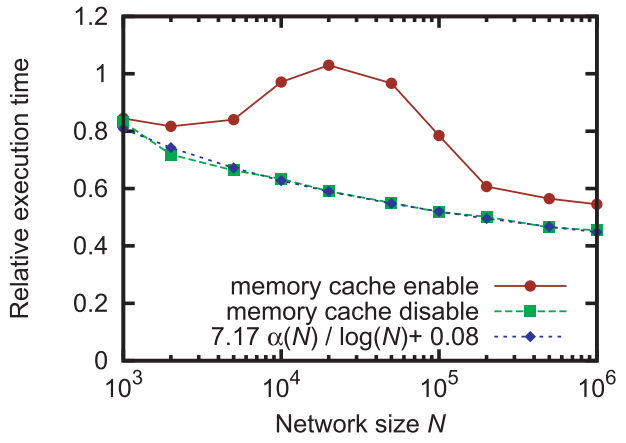

Fig. 8 Relative execution times with and without the memory cache for the average degree $k = 5$ and multiple edge weights of 1–1000.

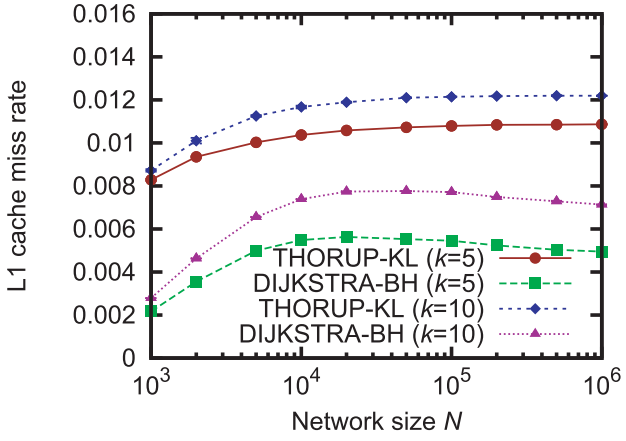

Fig. 9 L1 cache miss rate of THORUP-KL and DIJKSTRA-BH for the average degree $k = 5, 10$ and multiple edge weights of 1–1000.

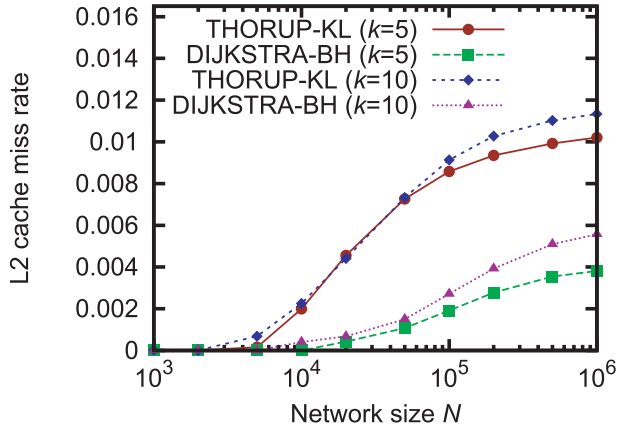

Fig. 10 L2 cache miss rate of THORUP-KL and DIJKSTRA-BH for the average degree $k = 5, 10$ and multiple edge weights of 1–1000.

ing the memory cache performance. We measured the memory cache performance (i.e., L1 and L2 cache miss rates) using a cache profiler called *cachegrind* [21]. Figures 9 and 10 show cache miss rates of THORUP-KL and DIJKSTRA-BH for L1 and L2 cache, respectively. These figures clearly indicate that THORUP-KL is not cache-friendly. More specifically, L1 and L2 cache miss rates of THORUP-KL are as approximately 2–4 times larger than those of DIJKSTRA-BH. Namely, THORUP-KL suffers much more frequently from cache miss penalties than DIJKSTRA-BH, leading to a significant slow down in program execution. From Fig. 10, we can explain why the relative execution time exhibits camel-shaped function. Figure 10 shows that the L2 cache miss rate of THORUP-KL increases more rapidly than that of DIJKSTRA-BH in medium-scale (several tens of thousands of network size) networks. Therefore, the relative execution time of Thorup's algorithm increases in medium-scale networks. Figure 8 shows that the L2 cache miss rates of THORUP-KL and DIJKSTRA-BH gently increase in large-scale (hundred thousands network size) networks. Therefore, in large-scale networks, the relative execution time of Thorup's algorithm decreases with the relative time complexity (Eq. (3)).

So, why is the THORUP-KL not cache-friendly? Memory cache performance is affected by several factors including the size of memory used and the locality of memory accesses (i.e., spatial and temporal localities) during program execution [22]. As explained in Sect. 3.3, both THORUP-KL and DIJKSTRA-BH are linear space algorithms. The inability of THORUP-KL to effectively use the cache compared with DIJKSTRA-BH can be explained as follows.

- Larger memory usage
  THORUP-KL consumes approximately 1.4 times more memory than DIJKSTRA-BH (see Fig. 7). Larger memory consumption results in a higher cache miss rate.
- Lower spatial locality
  The hierarchical bucketing structure of THORUP-KL is sparsely allocated in memory while the binary heap of DIJKSTRA-BH is densely allocated. The hierarchical bucketing structure consists of several types of elements such as references (i.e., pointers) to buckets and references to child components. The binary heap can be realized as an array. The lower spatial locality of THORUP-KL results in a higher cache miss rate.
- Lower temporal locality
  THORUP-KL needs recursion, whereas DIJK-STRA-BH doesn't. Because it uses the hierarchical bucketing structure, THORUP-KL needs to recursively visit components. Recursion generally results in lower temporal locality. On the contrary, DIJKSTRA-BH requires only looping. Lower temporal locality of THORUP-KL results in a higher cache miss rate.

Therefore, as the camel-shaped function of the relative execution time indicates, its performance is significantly impacted by the memory cache performance.

Table 1 Measured CPI time, access time and cache miss penalty.

CPI (Cycles Per Instruction) time	T_{CPU}	0.96	[ns]
access time of L1 cache	T_{AL1}	1.4	[ns]
cache miss penalty of L1 cache	T_{L1}	49.4	[ns]
cache miss penalty of L2 cache	T_{L2}	435.3	[ns]

Table 2 System specifications.

FSB (Front-Side Bus) clock	800	[MHz]
L1 cache size	32	[kbyte]
L2 cache size	1024	[kbyte]
memory	DDR2-533	
memory controller hub	Intel 82945G	
memory clock	133	[MHz]
memory bus width	128	[bit]
I/O bus clock	266	[MHz]

3.5 Understanding Effect of Memory Cache Performance on Relative Execution Time

For practically utilizing THORUP-KL for network simulations, a thorough understanding of its performance is necessary. In particular, we need to understand how the performance of THORUP-KL is affected by memory cache performance. Hence, we finally try to answer the last question: how is the relative execution time affected by several factors such as L1 and L2 cache miss rates and their cache miss penalties?

Let us introduce a simple cache performance model [23], which approximates the execution time $\tilde{T}$ of a program on a microprocessor with L1 and L2 memory caches.

$$\tilde{T} = N_I\, T_{CPU} + N_M(T_{AL1} + p_{L1}\, T_{L1} + p_{L2}\, T_{L2}) \qquad (4)$$

In the above equation, N_I is the number of instructions executed, T_{CPU} is the CPI (Cycles Per Instruction) time, and N_M is the number of memory accesses performed. Also, T_{AL1} is the access time of L1 cache, and T_{Li} and p_{Li} $(i = 1, 2)$ are the cache miss penalty and Li cache miss rate, respectively.

We already have L1 and L2 cache miss rates, p_{L1} and p_{L2}, in Figs. 9 and 10. The number of instructions executed, N_I, and the number of memory accesses performed, N_M, can be obtained with cachegrind. We then measured other system-dependent (and program-independent) parameters, T_{CPU}, T_{AL1}, T_{L1}, and T_{L2}, in the computer with system specification shown in Table 2 using a benchmark tool called *lmbench* [24]. Our measurement results are summarized in Table 1.

By setting all parameters in Eq. (4) to our measured values, execution times of THORUP-KL and DIJKSTRA-BH, $\tilde{T}_T$ and $\tilde{T}_D$, can be estimated. Thus, the estimated relative execution time, $\tilde{T}_T/\tilde{T}_D$, can be measured. Figure 11 shows the estimated relative execution times obtained from Eq. (4) as well as the relative execution time obtained in Sect. 3.2.

Figure 11 shows that the simple cache performance model well explains the camel-shaped function of the relative execution time. It should be noted that both the relative execution time and the estimated relative execution time take the maximum value at network size between 10^4 and 10^5 (see Fig. 11). Namely, the simple cache performance model roughly captures the dynamics of the relative execution time. In other words, the camel-shaped function of the relative execution time can be explained solely by the memory cache performance.

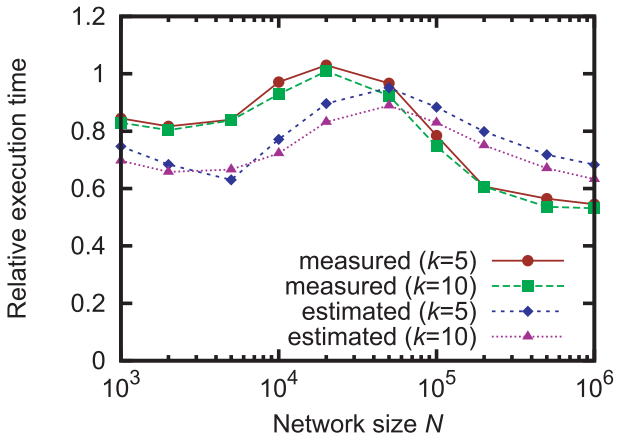

Fig. 11 Measured and estimated relative execution times with the memory cache for the average degrees $k = 5, 10$ and multiple edge weights of 1–1000.

3.6 Discussion

As explained in Sect. 2, the time complexity of THORUP-FR is $O(M+N)$ whereas that of THORUP-KL is $O(M\,\alpha(N)+N)$. We intentionally used THORUP-KL instead of THORUP-FR. Our experiments clearly show that the practical efficiency of an algorithm cannot be predicted from just its time complexity. Its performance is significantly affected by how well it works with memory cache.

In Sects. 3.4 and 3.5, we investigated the effect of memory cache on performance of THORUP-KL and DIJKSTRA-BH from two different approaches. It is well known that performance of an algorithm is considerably influenced by memory cache [22], [25], [26]. Also, studies on *cache-aware* and *cache-oblivious* algorithms have been actively performed [25], [27], [28]. The effect of memory cache on the performance of an algorithm is quite complicated. Therefore, its performance should be carefully investigated with extensive experiments, as we have done in this paper.

In Sect. 3.2, we have shown that THORUP-KL is almost always faster than DIJKSTRA-BH regardless of the network size N, the average degree k, and the type of edge weights. This implies that THORUP-KL is superior to DIJKSTRA-BH for network simulations. However, as discussed in Sect. 3.1, we have only used a simple network model (i.e., random network). The performance of THORUP-KL and DIJKSTRA-BH might be affected by the type of networks (e.g., hierarchical network [29] and scale-free network [30]). More investigation with realistic network models would be of great interest.

In Sect. 3.5, we demonstrated the potential of a simple

cache performance model for analyzing the effect of system performance on the execution time. For instance, the simple model enables us to predict the effect of system performance improvement/degradation on the execution time. More specifically, the execution time with a double-speed microprocessor can be predicted simply by halving T_{CPU} in Eq. (4). Also, the execution time with double-speed L1 cache memory can be predicted simply by halving T_{L1} in Eq. (4). Note that program-specific parameters (i.e., the number of instructions executed, N_I, the number of memory accesses performed, N_M, and cache miss rates p_{L1} and p_{L2}) are independent of other system-specific parameters (i.e., T_{CPU}, T_{L1} and T_{L2}). Thus, we can freely change system parameters in Eq. (4) so as to estimate their effect on the execution time. We are planning to investigate the effect of system architecture (e.g., type of microprocessors and memory cache architecture) on the performance of THORUP-KL and DIJKSTRA-BH.

It is beyond the scope of this paper, but the performance of THORUP-KL and DIJKSTRA-BH in a parallel environment should be investigated since the usage of SMP processors and/or parallel computers might be the only viable choice for very large-scale network simulations. Multiple instances of a single-source shortest path algorithm can be executed in parallel on SMP processors and/or parallel computers.

4. Conclusion

This paper investigated the performance of Thorup's algorithm by comparing it to Dijkstra's, and answering the following questions.

1. How efficiently/inefficiently does Thorup's algorithm perform compared with Dijkstra's for large-scale (e.g., million nodes) network simulations?
2. How and why does the practical performance of Thorup's and Dijkstra's algorithms deviate from their time complexities (i.e., theoretical performance)?

In this paper, we intentionally used THORUP-KL (i.e., a *simplified* version of Thorup's algorithm) with the time complexity of $O(M\,\alpha(N) + N)$.

There are several variants of Dijkstra's algorithm with different time complexities of $O(M + N^2)$ [12], $O((M + N)\log N)$ [5] and $O(M + N\log N)$ [11]. In this paper, we focused on Dijkstra's algorithm with a binary heap (DIJKSTRA-BH) [5] with the time complexity of $O((M + N)\log N)$.

Extensive experiments on our implementations of THORUP-KL and DIJKSTRA-BH yielded a performance comparison in terms of execution time and memory consumption. Our findings include (1) THORUP-KL is almost always faster than DIJKSTRA-BH for large-scale network simulations, and (2) the performances of THORUP-KL and DIJKSTRA-BH deviate from their time complexities if they access memory cache in the microprocessor.

Acknowledgement

This work was supported by the Grant-in-Aid for Young Scientists (A) (19680003) by the Ministry of Education, Culture, Sports, Science and Technology (MEXT).

References

[1] D.M. Nicol, M. Liljenstam, and J. Liu, "Advanced concepts in large-scale network simulation," Proc. 37th Conference on Winter Simulation, pp.153–166, Dec. 2005.
[2] P. Huang and J. Heidemann, "Minimizing routing state for light-weight network simulation," Proc. IEEE Computer Society's 9th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS), pp.108–116, Aug. 2001.
[3] Z. Hao, X. Yun, and H. Zhang, "An efficient routing mechanism in network simulation," Simulation, vol.84, no.10-11, pp.511–520, May 2008.
[4] "The network simulator — ns2.2 available at http://www.isi.edu/nsnam/ns/
[5] J. Williams, "Heapsort," Commun. ACM, vol.7, no.6, pp.347–348, 1964.
[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed., MIT Press, 2009.
[7] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," J. ACM (JACM), vol.46, no.3, pp.362–394, May 1999.
[8] M.L. Fredman and D.E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," J. Comput. Syst. Sci., vol.48, no.3, pp.533–551, June 1994.
[9] J.B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," Proc. American Mathematical Society, pp.48–50, JSTOR, Feb. 1956.
[10] Y. Asano and H. Imai, "Practical efficiency of the linear-time algorithm for the single source shortest path problem," J. Operations Research Society of Japan, vol.43, no.4, pp.431–447, Dec. 2000.
[11] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," J. ACM (JACM), vol.34, no.3, pp.596–615, July 1987.
[12] E.W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, vol.1, no.1, pp.269–271, Dec. 1959.
[13] B.V. Cherkassky, A.V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," Mathematical Programming, vol.73, no.2, pp.129–174, May 1996.
[14] A.V. Goldberg and R.E. Tarjan, "Expected performance of Dijkstra's shortest path algorithm," NEC research, Technical Report, 96-062, June 1996.
[15] J.A. Storer, An introduction to data structures and algorithms, Birkhäuser Boston, Nov. 2001.
[16] B. Bloolbas, Random Graphs Second ed., Cambridge University Press, 2001.
[17] H. Tangmunarunkfit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topology generators: Degree-based vs. structural," ACM SIGCOMM Computer Communication Review, vol.32, no.4, pp.147–159, Oct. 2002.
[18] "CAIDA — skitter." http://www.caida.org/tools/measurement/skitter/
[19] P. Mahadevan, C. Hubble, D. Krioukov, B. Huffaker, and A. Vahdat, "Orbis: Rescaling degree correlations to generate annotated internet topologies," ACM SIGCOMM Computer Communication Review, vol.37, no.4, pp.325–336, Oct. 2007.
[20] L. Fortnow and S. Homer, "A short history of computational complexity," Bulletin of the European Association for Theoretical Computer Science Computational Complexity Column, vol.80, pp.95–133, June 2003.

[21] "Cachegrind — a cache profiler," http://valgrind.org/info/tools.html
[22] J. Handy, The cache memory book, Morgan Kaufmann, Jan. 1998.
[23] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 3rd ed., Morgan Kaufmann, May 2002.
[24] "Lmbench — Tools for performance analysis," http://www.bitmover.com/lmbench/
[25] R.E. Ladner, R. Fortna, and B.H. Nguyen, "A comparison of cache aware and cache oblivious static search trees using program instrumentation," Experimental Algorithmics, pp.78–92, Jan. 2002.
[26] A. LaMarca and R.E. Ladner, "The influence of caches on the performance of sorting," Proc. Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pp.370–379, Jan. 1997.
[27] G.S. Brodal, R. Fagerberg, and R. Jacob, "Cache oblivious search trees via binary trees of small height," Proc. Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp.39–48, Jan. 2002.
[28] L. Arge, M.A. Bender, E.D. Demaine, B. Holland-Minkley, and J.I. Munro, "Cache-oblivious priority queue and graph algorithm applications," Proc. Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp.268–276, May 2002.
[29] K. Calvert, M.B. Doar, A. Nexion, E.W. Zegura, G. Tech, and G. Tech, "Modeling internet topology," IEEE Commun. Mag., vol.35, no.6, pp.160–163, June 1997.
[30] A.L. Bárabasi and R. Albert, "Emergence of scaling in random networks," Science, vol.286, no.5439, pp.509–512, Oct. 1999.

Yusuke Sakumoto received M.E. and Ph.D. degrees in the Information and Computer Sciences from Osaka University in 2008 and 2010, respectively. He is currently an assistant professor at Faculty of System Design, Tokyo Metropolitan University, Japan. His research work is in the area of performance evaluation of congestion control protocol. He is a member of IEEE and IPSJ.

Hiroyuki Ohsaki received the M.E. degree in the Information and Computer Sciences from Osaka University, Osaka, Japan, in 1995. He also received the Ph.D. degree from Osaka University, Osaka, Japan, in 1997. He is currently an associate professor at Department of Information Networking, Graduate School of Information Science and Technology, Osaka University, Japan. His research work is in the area of traffic management in high-speed networks. He is a member of IEEE and IPSJ.

Makoto Imase received his B.E. and M.E. degrees in information engineering from Osaka University in 1975 and 1977, respectively. He received D.E. degree from Osaka University in 1986. From 1977 to 2001, he was engaged Nippon Telegraph and Telephone Corporation (NTT). He has been a Professor of Graduate School of Information Science and Technology at Osaka University since 2002. His research interests are in the area of information networks, distributed systems and graph theory. He is a member of IPSJ and JSIAM.