Detouring Skip Graph: A Structured Overlay Utilizing Detour Routes

Takeshi Kaneko*¶, Ryohei Banno*, Kazuyuki Shudo*, Yusuke Aoki*, Kota Abe†‡ and Yuuichi Teranishi‡§
* Tokyo Institute of Technology, Tokyo, Japan
† Osaka City University, Osaka, Japan
‡ National Institute of Information and Communications Technology, Tokyo, Japan
§ Osaka University, Osaka, Japan
Email: ¶kaneko.t.ay@m.titech.ac.jp

***Abstract*—Skip Graph, one of the structured overlays, provides a scalable network owing to the routing path lengths of $O(\log n)$, where n denotes the total number of nodes. However, there is a problem that most of the routing paths are quite longer than the shortest paths because each node in the network knows only its neighbors, rather than the global topology. In general, long routing paths lead to long delay times and low fault tolerance. Herein, we propose Detouring Skip Graph, which shortens the path lengths through the use of detour routes. It does not require construction of extra links or modification of its topology; thereby, it can succeed in shortening them while maintaining the advantages of Skip Graph. The evaluation experiments show that the average path length was shortened by approximately 20%–30% in comparison with Skip Graph.**

***Index Terms*—Skip Graph, structured overlay, routing algorithm, detour route**

I. Introduction

Overlay networks are application-level logical networks built on existing networks such as the Internet. Specially structured overlays construct autonomous distributed networks according to specific data structures or protocols; thereby providing reachability to target nodes, high scalability, efficient routings, and high fault tolerance. Owing to the properties, application to large-scale distributed systems such as distributed key/value stores [1], video streaming [2], and online games [3] has been proposed. In recent years, application to the fields of IoT and Blockchain is also expected [4], [5].

Skip Graph [6], one of the structured overlays, is a distributed data structure that provides the capability of range queries as a result of preserving the order of keys by managing data without hashing. Over the years, numerous extensions and improvements of Skip Graph have been proposed, even in recent studies [7], [8].

However, it is still a challenge that each node cannot take full advantage of existing links because it knows only its neighbors, rather than the global topology. Thus, the routing paths tend to be quite longer than the shortest paths. In general, overlay networks whose routing paths are long lead to long delay times and low fault tolerance. Since most application mentioned above of overlay networks requires the responsiveness and the reliability, shortening routing paths is a critical demand for overlays including Skip Graph.

This work was supported by JSPS KAKENHI Grant Numbers 19K20253, New Energy and Industrial Technology Development Organization (NEDO), and SECOM Science and Technology Foundation.

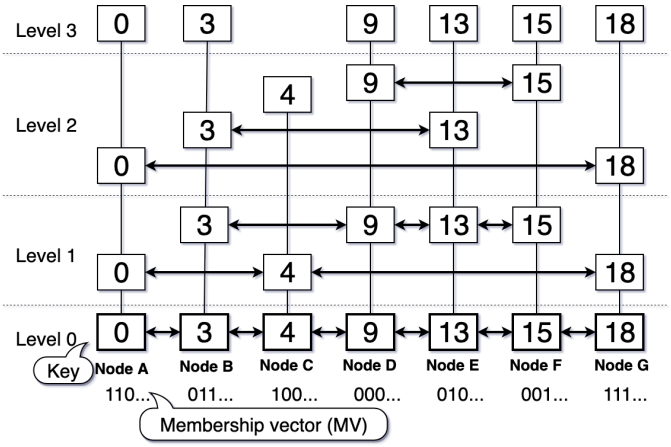

Fig. 1: An example of Skip Graph.

Herein, we propose an extension of Skip Graph which is called Detouring Skip Graph. It shortens the path lengths through the more efficient use of existing links while maintaining the advantages of Skip Graph. Specifically, its routing algorithm is different from that of Skip Graph in two ways: each node 1) utilizes detour routes and 2) traverses adjacent nodes from its maximum level.

The rest of this paper is organized as follows. Section II presents an overview of Skip Graph and the related work on shortening path lengths. Section III presents Detouring Skip Graph in detail. Section IV presents the evaluation experiments for the proposed method and the results. Finally, Section V presents the conclusion of this study.

II. Related Work

A. Skip Graph

Skip Graph is a distributed data structure designed based on Skip List [9], and each node belongs to multiple sorted doubly linked lists. Figure 1 shows an example of a topology of Skip Graph. Each node has a key in a totally ordered set and a random string called membership vector (MV), which

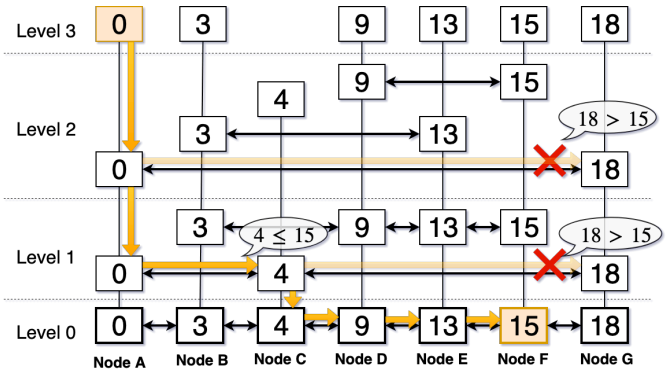

Fig. 2: A `searchOp` routing from node A to key 15.

plays a key role in constructing the topology of Skip Graph. In the figure, the set of alphabets that are elements of MV is $\{0, 1\}$. In a linked list at level l, the leading l digits of the MV of every node is the same as that of all others. Particularly at level 0, all nodes belong to one linked list. Therefore, each node belongs to $O(\log n)$ linked lists. Further, by using the same method as Skip List, Skip Graph achieves a routing path length of $O(\log n)$ for a query to search a key k_{target}. A detailed explanation of the routing algorithm is presented below, where it is assumed that the keys in the linked lists are sorted in ascending order from left to right.

Herein, suppose node $v_{current}$ in Skip Graph is receiving a query `searchOp` to search a node that has a specific key. The query has three information $(v_{start}, k_{target}, l_{prev})$: a start node v_{start}, a target key k_{target}, and the level l_{prev} at which the previous node sends the query. If $v_{current}.key$, which means the key of $v_{current}$, equals to k_{target}, $v_{current}$ sends a query `foundOp` to v_{start} since it means that $v_{current}$ is the target node. If $v_{current}.key < k_{target}$, $v_{current}$ traverses the right adjacent nodes at the levels from l_{prev} in descending order and sends a search query `searchOp` to the first adjacent node v_{next} where $v_{next}.key \leq k_{target}$. If $v_{current}.key > k_{target}$, $v_{current}$ traverses the left adjacent nodes and sends a search query `searchOp` to the next node in a similar manner.

Figure 2 shows a routing process for a query to target key $k_{target} = 15$ issued at node A on the condition that each node follows the above method in the topology shown in Figure 1. Then, the key sequence of the nodes on the path is $(0, 4, 9, 13, 15)$, and the path length is 4.

B. Shortening Path Lengths

The path length is 4 in Figure 2, however, there are shorter paths in reality. For instance, if node A chooses node G at level 2 instead of node C at level 1 as the next node, the key sequence of the nodes on the path would be $(0, 18, 15)$, and the path length would be 2 (which is shorter than 4). Thus, the routing of Skip Graph is inefficient in that it cannot fully utilize the existing links. Therefore, various approaches have been proposed for shortening the path lengths.

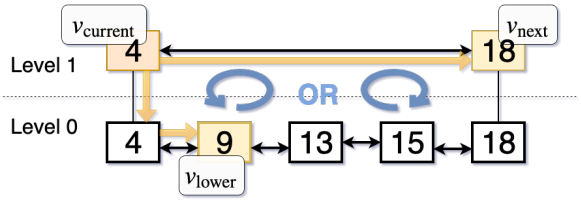

Fig. 3: Should node $v_{current}$ select the detour route?

In the above example, at the route from node A to node G, the magnitude relationship between the key of the node and the target key k_{target} is reversed. Routes like this are hereinafter referred to as "detour routes." The proposed method (described later in detail) utilizes detour routes. It is similar to the method proposed by Higuchi et al. [10] in terms of use of detour routes. However, the subject of their method is not ordinary Skip Graph but Skip Graph whose topology is balanced by means of using linear hashing preserving the order.

There are already several methods to shorten path lengths of Skip Graph routings: e.g., methods that involve construction of extra links in the Skip Graph and appropriate routings on the topology [11], [12], and methods that reconstruct or refine the unbalanced topology resulting from randomly generated MVs into the ideal topology [7], [8], [13], [14]. However, the construction of extra links and the modification of the topology lead to an increase in necessary transfer messages and maintenance cost.

III. Proposed Method

The main idea of Detouring Skip Graph is the utilization of detour routes. Moreover, the idea can be combined with the technique traversing from maximum levels. This section presents these ideas and the details of Detouring Skip Graph whose routing algorithm combines them.

In the following, let K be a set of keys. To simplify, we assume that K is a subset of rational numbers[1] and multiple nodes do not have the same key. Thus, arithmetic operations and absolute values are well-defined on K.

A. Utilizing Detour Routes

The routing algorithm of Detouring Skip Graph utilizes detour routes. The idea is based on the following argument.

Figure 3 shows a part of the topology of Figure 1. Let v_{next} and v_{lower} be the right neighbors of node $v_{current}$ at level 1 and 0, respectively. Now, suppose $v_{current}$ is receiving a query to target key k_{target} where $9 \leq k_{target} < 18$. In a situation where it follows the routing algorithm of Skip Graph, it selects v_{lower} as the next node. If $k_{target} = 15$, the key sequence of the nodes on the path would be $(4, 9, 13, 15)$, and the path length would be 3. However, in a situation where it selects v_{next} as the next node, i.e., it uses the detour route, the key sequence would be $(4, 18, 15)$, and the path length would be 2. The path length of the latter is shorter than that of the former. The effectiveness

[1]As a mathematical fact, any countable totally ordered set can be order embedded into the set of rational numbers [15].

of such detour routes is dependent on the position of the target key; if $k_{target} = 13$, the path length of using the detour route would be longer than that of using the ordinary route.

As shown in Figure 3, it can be determined from the center of the node sequence at the lower level (level 0 in Figure 3) whether $v_{current}$ should select v_{next} or v_{lower} to shorten the path length. This section presents the routing algorithm that each node determines the next node based on a detour criterion at each level.

Algorithm 1: `searchDROp` in node $v_{current}$

```
/* utilizing Detour Routes                                          */
upon receiving ⟨searchDROp, v_start, k_target, l_prev⟩ then
    if v_current.key = k_target then
        send ⟨foundOp, v_current⟩ to v_start ;
        return;
    else if v_current.key < k_target then
        for l_current ← l_prev downTo 0 do
            v_next ← v_current.neighbors[R][l_current];
            if v_next.key ≤ k_target then
                send ⟨searchDROp, v_start, k_target, l_current⟩ to
                    v_next ;
                return;
            else if l_current > 0 then
                v_lower ← v_current.neighbors[R][l_current − 1];
                if closeToRight(k_target, v_lower.key, v_next.key) then
                    send ⟨searchDROp, v_start, k_target, l_current⟩ to
                        v_next ;
                    return;
    else
        for l_current ← l_prev downTo 0 do
            v_next ← v_current.neighbors[L][l_current];
            if v_next.key ≥ k_target then
                send ⟨searchDROp, v_start, k_target, l_current⟩ to
                    v_next ;
                return;
            else if l_current > 0 then
                v_lower ← v_current.neighbors[L][l_current − 1];
                if ¬ closeToRight(k_target, v_next.key, v_lower.key) then
                    send ⟨searchDROp, v_start, k_target, l_current⟩ to
                        v_next ;
                    return;
    send ⟨notFoundOp, v_current⟩ to v_start;
function closeToRight(k_target, k_left, k_right)
    k_mid ← mid(k_left, k_right);
    return k_mid < k_target;
```

Algorithm 1 is a pseudocode of this algorithm. Herein, $v_{current}$ traverses adjacent nodes in the same way as Skip Graph when node $v_{current}$ receives a query `searchDROp`. However, if $v_{current}$ judges that using a detour route is better than not using it, $v_{current}$ selects the end node of the detour route as the next node. Function $closeToRight(k_{target}, k_{left}, k_{right})$ can be used to make this judgment. Let I_{right} be $\{k \in K \mid k \geq k_{right}\}$. The function returns *true* if the signed distance from k_{target} to I_{right} is smaller than that from $mid(k_{left}, k_{right})$ to I_{right}, i.e., $k_{right} - k_{target} < k_{right} - mid(k_{left}, k_{right})$; otherwise, it returns *false*. Intuitively, it means that k_{target} is closer to k_{right} than

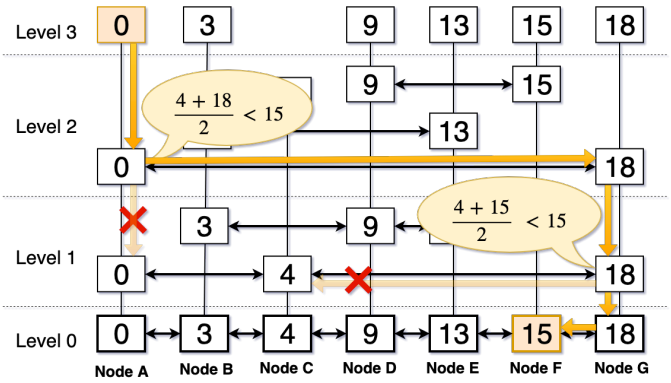

Fig. 4: A `searchDROp` routing from node A to key 15 where $mid(k_1, k_2) = \frac{k_1+k_2}{2}$.

$mid(k_{left}, k_{right})$. Further, *mid* is a design parameter that can be defined as a function before the construction of the topology where it satisfies that the following 1) and 2) are approximately equal for the set V of all participating nodes at any point in time and any $k_1, k_2 \in K$ $(k_1 \leq k_2)$.

1) $\#\left\{k \in K \mid k_1 \leq k \leq mid(k_1, k_2) \wedge^{\exists} v \in V, v.key = k\right\}$
2) $\#\left\{k \in K \mid mid(k_1, k_2) \leq k \leq k_2 \wedge^{\exists} v \in V, v.key = k\right\}$

We give some examples of defining *mid*. Let $v.key$, the key of a node v, be regarded as a random variable. If

$$K = \{0, 1, \ldots, n-1\} \text{ and } P\{v.key = k\} = \frac{1}{n},$$

then

$$mid(k_1, k_2) := \frac{k_1 + k_2}{2}.$$

If

$$K = \{0, 1, \ldots\} \text{ and } P\{v.key = k\} = \left(\frac{1}{2}\right)^{k+1},$$

then

$$mid(k_1, k_2) := -\log_2 \left(\frac{2\left(\frac{1}{2}\right)^{k_1} + \left(\frac{1}{2}\right)^{k_2}}{3}\right).$$

By defining *mid* in this way, $mid(v_{lower}.key, v_{next}.key)$ or $mid(v_{next}.key, v_{prev}.key)$ refers to key estimation of the center of the lower-level node sequence. Thus, *closeToRight* plays the appropriate role of a detour judgment.

In practice, it is difficult to obtain the distribution of the keys beforehand. However, from the evaluation experiment presented in Section IV, we observed that it is effective in many cases for shortening path lengths by defining *mid* as:

$$mid(k_1, k_2) := \frac{k_1 + k_2}{2}.$$

Figure 4 shows the routing process for a query to target key $k_{target} = 15$ issued at node A on the condition that each node follows Algorithm 1 in the topology shown in Figure 1. The key sequence of the nodes on the path is $(0, 18, 15)$, and the path length is 2, which is shorter than that of Figure 2.

B. Traversing from the Maximum Level

In addition to utilizing detour routes, we can improve the routings of Skip Graph by making each node to traverse from the maximum level.

As discussed in Section II-A, in the routings of Skip Graph, node $v_{current}$ traverses the adjacent nodes at the levels from reception level l_{prev} in descending order and determines the first adjacent node v_{next} that satisfies the condition as the next node. The levels are monotonically decreasing for the entire routing. However, there are cases where adjacent nodes at levels larger than l_{prev} satisfy the condition. Moreover, the larger the sending level, the larger the difference in key between adjacent nodes. Therefore, the difference between the key of the next node and the target key k_{target} is not larger when traversing from level $v_{current}.maxLevel$ than when traversing from level l_{prev}, where $v.maxLevel$ is the maximum level of node v. Thus, it is effective in shortening the path lengths that each node traverses from its maximum level.

Algorithm 2: `searchMLOp` in node $v_{current}$

```
   /* traversing from Max Level                                   */
1  upon receiving ⟨searchMLOp, v_start, k_target⟩ then
2      if v_current.key = k_target then
3          send ⟨foundOp, v_current⟩ to v_start ;
4          return;
5      else if v_current.key < k_target then
6          for l_current ← v_current.maxLevel downTo 0 do
7              v_next ← v_current.neighbors[R][l_current];
8              if v_next.key ≤ k_target then
9                  send ⟨searchMLOp, v_start, k_target⟩ to v_next ;
10                 return;
11     else
12         for l_current ← v_current.maxLevel downTo 0 do
13             v_next ← v_current.neighbors[L][l_current];
14             if v_next.key ≥ k_target then
15                 send ⟨searchMLOp, v_start, k_target⟩ to v_next ;
16                 return;
17     send ⟨notFoundOp, v_current⟩ to v_start
```

Algorithm 2 is a pseudocode of this algorithm. When node $v_{current}$ receives a query `searchMLOp`, $v_{current}$ traverses adjacent nodes from $v_{current}.maxLevel$ and sends a query `searchMLOp` to the first node that satisfies the condition. This routing differs from that of Skip Graph only in the start level of traversing. Note that it is possible to use binary search for finding a next node instead of linear search, which is faster, but this code uses the latter for simplicity.

Figure 5 shows a routing process for a query to target key $k_{target} = 15$ issued at node A on the condition that each node follows Algorithm 2 in the topology shown in Figure 1. The key sequence of the nodes on the path is $(0, 4, 9, 15)$, and the path length is 3, which is shorter than that of Figure 2.

It should be noted that Algorithm 2 has the disadvantage of increasing the computation costs incurred between receiving a query and determining the next node, although it has the advantage of shortening the path lengths. Let l_{MAX} be the maximum value of the maximum levels of all nodes, which is $O(\log n)$, and let H be the path length, which is $O(\log n)$. Then, the sum of the time required for each node on a routing path to determine the next node until finishing a rounting process, except for the communication time and the I/O processing time, is $O(l_{MAX} + H) = O(\log n)$ for Skip Graph. On the other hand, in the case of the following Algorithm 2, it is $O(H \cdot l_{MAX}) = O(\log^2 n)$. Especially when using binary search, it is $O(H \log l_{MAX}) = O(\log n \cdot \log(\log n))$. These are inferior to Skip Graph in terms of computational complexity. However, the time taken for a routing is typically dominated by the communication time, hence it is more important to shorten path lengths in most cases.

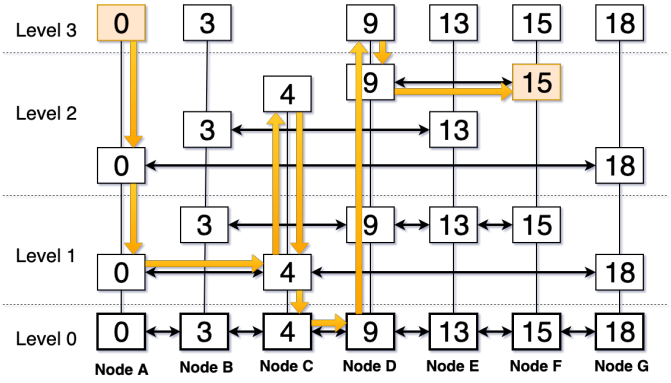

Fig. 5: A `searchMLOp` routing from node A to key 15.

C. Detouring Skip Graph: Combining Two Improvements

Because the two improved routing algorithms described above are independent changes from the routing algorithm of Skip Graph, an algorithm combining them can be defined naturally. We refer an extension of Skip Graph that performs such routing as Detouring Skip Graph.

When node $v_{current}$ receives a query `searchDSGOp`, $v_{current}$ traverses adjacent nodes from $v_{current}.maxLevel$ in the same way as described in Section III-B and sends a query `searchDSGOp` to the first node that satisfies the condition. In the process, `searchDSGOp` uses detour routes based on the detour judgment as described in Section III-A.

From the foregoing, Detouring Skip Graph can bring about the shortening of path lengths for search queries. Unlike the existing methods discussed in Section II-B, it does not require construction of extra links or modification of its topology. Therefore, there is no increase in message transfer and management costs. Additionally, it is an extension that maintains the good properties of Skip Graph.

D. Reachability of Detouring Skip Graph

Detouring Skip Graph is characterized in that the key sequences of the nodes on a routing path are not monotonic in order, which is a property that does not apply to Skip Graph. You might consider that the routing process may lead to an infinite loop. However, the reachability is guaranteed, which

is proved in this section. It should be assumed that Detouring Skip Graph does not separate as a network.

Suppose a search query whose target key is k is being issued. Then, each variable can be defined as follows.

- Let $(a_1, a_2, \ldots)$ be the key sequence of the nodes on the routing path.
- $S_k := \{x \in K \mid x < k\}$.
- $T_k := \{x \in K \mid x > k\}$.
- Let $(a_{s_1}, a_{s_2}, \ldots)$ be the subsequence of $(a_i)_i$, whose elements are all a_i satisfying $a_i \in S_k$.
- Let $(a_{t_1}, a_{t_2}, \ldots)$ be the subsequence of $(a_i)_i$, whose elements are all a_i satisfying $a_i \in T_k$.
- Let binary relation $\succ_k$, $\prec_k$ on $S_k \times T_k$ be:
  - $\succ_k := \{(x, y) \in S_k \times T_k \mid \textit{closeToRight}(k, x, y)\}$
  - $\prec_k := \{(x, y) \in S_k \times T_k \mid \neg\textit{closeToRight}(k, x, y)\}$.

  Intuitively, $x \succ_k y$ implies that y is closer to k than x, and $x \prec_k y$ implies that x is closer to k than y.

The query reaches the target node if and only if $(a_i)_i$ is a finite sequence. Thus, it is sufficient to show that $(a_i)_i$ is finite. If both $(a_{s_j})_j$ and $(a_{t_j})_j$ are strictly approaching k (i.e., they are strictly increasing and strictly decreasing, respectively), $(a_i)_i$ converges to the key of the target node in finite steps and $(a_i)_i$ is a finite sequence.

Next, to show the strict monotonicity of each subsequence, the function *mid* used as the detour judgment must exhibit the following property.

Property 1: $^\forall x \in S_k, ^\forall y \in T_k$,

- $x \prec_k y \Rightarrow \begin{cases} ^\forall x' \in S_k, & [x' \le x \Rightarrow x' \prec_k y] \\ ^\forall y' \in T_k, & [y' \le y \Rightarrow x \prec_k y'] \end{cases}$
- $x \succ_k y \Rightarrow \begin{cases} ^\forall x' \in S_k, & [x' \ge x \Rightarrow x' \succ_k y] \\ ^\forall y' \in T_k, & [y' \ge y \Rightarrow x \succ_k y'] \end{cases}$

Property 1 is satisfied whenever *mid* is defined as the median estimation based on any probability distribution, e.g., $\textit{mid}(x, y) := \frac{x+y}{2}$. Then, the following lemma holds.

Lemma 1: $^\forall x_1, x_2 \in S_k, ^\forall y_1, y_2 \in T_k$,

- $\left[^\exists x \in S_k \text{ s.t. } (x \prec_k y_1 \wedge x \succ_k y_2)\right] \Rightarrow y_1 < y_2$
- $\left[^\exists y \in T_k \text{ s.t. } (x_1 \succ_k y \wedge x_2 \prec_k y)\right] \Rightarrow x_1 > x_2$

Proof: Suppose there exists $x \in S_k$ such that $x \prec_k y_1$ and $x \succ_k y_2$. If $y_1 \ge y_2$, then $x \succ_k y_1$ because of $x \succ_k y_2$ and Property 1, however it contradicts $x \prec_k y_1$. Therefore, we have that $y_1 < y_2$. The latter proposition can also be established in the same way. ■

Lemma 1 derives the following theorem.

Theorem 1: $(a_{s_j})_j$ and $(a_{t_j})_j$ are strictly increasing and strictly decreasing, respectively.

Proof: It is sufficient for each step i to show that:

$$\begin{cases} j > 1 \Rightarrow a_{s_j} > a_{s_{j-1}} & (\text{ if } ^\exists j \text{ s.t. } s_j = i) \\ j > 1 \Rightarrow a_{t_j} < a_{t_{j-1}} & (\text{ if } ^\exists j \text{ s.t. } t_j = i), \end{cases} \quad (*)$$

where step i represents the process on the i-th node in the routing path. This can be shown using mathematical induction.

Suppose $(*)$ holds at step $1, 2, \ldots, i$.

1) If $a_i = k$, then step $i+1$ does not exist because the routing process is complete.
2) If $a_i \in S_k$, then the four cases are considered: (i) $a_{i+1} \in S_k$, (ii) $a_{i+1} \in T_k$, (iii) $a_{i+1} = k$, and (iv) a_{i+1} does not exist. In case (i) and (iii), $(*)$ holds at step $i+1$ because $a_i < a_{i+1}$ and $a_{i+1} \notin S_k \cup T_k$, respectively. In case (iv), step $i+1$ does not exist because the routing process is complete. In case (ii), because a detour route is used, there exist j and $x \in S_k$ such that $t_j = i+1$, $x > a_i$, and $x \prec_k a_{t_j}$. From Property 1, we have $a_{t_{j-1}+1} \succ_k a_{t_{j-1}}$. If $j > 1$, then it implies that a detour route was used at step t_{j-1} and that no detour route was used at step $t_{j-1}+1, t_{j-1}+2, \ldots, i-1$ owing to the definition of subsequence $(a_{t_{j'}})_{j'}$. Thus, $a_{t_{j-1}+1}, a_{t_{j-1}+2}, \ldots, a_i \in S_k$, and there exists $y \in T_k$ such that $y < a_{t_{j-1}}$ and $a_{t_{j-1}+1} \succ_k y$. From Property 1, we have $a_{t_{j-1}+1} \succ_k a_{t_{j-1}}$. In addition, $a_{t_{j-1}+1} < a_{t_{j-1}+2} < \cdots < a_i$ holds by the induction hypothesis. From Property 1, we have $a_i \succ_k a_{t_{j-1}}$. Thus, $a_{t_j} < a_{t_{j-1}}$ holds because of Lemma 1, i.e., $(*)$ holds at step $i+1$.
3) If $a_i \in T_k$, then $(*)$ holds at step $i+1$, which can be shown in the same way as 2).

■

Therefore, the reachability to the target node is guaranteed for any search query.

TABLE I: Routing algorithms used as the evaluation subjects.

`searchOp`	: Skip Graph	(Sec. II-A)
`searchDROp`	: Utilizing detour routes	(Sec. III-A)
`searchMLOp`	: Traversing from max level	(Sec. III-B)
`searchDSGOp`	: Detouring Skip Graph	(Sec. III-C)

IV. Evaluation

We evaluated the path length by conducting a simulation experiment and observed the effect of the proposed method. We experimented with the following three key generation methods:

- Generated by uniform distribution.
- Generated by power-law distribution.
- Random English titles on Wikipedia.

A. Generated by Uniform Distribution

We used pseudorandom numbers to generate keys so that the keys of participating nodes follows a uniform distribution $P\{v.\textit{key} = k\} = \frac{1}{2^{30}}$ $(k \in \{0, 1, \ldots, 2^{30}-1\})$ where $v.\textit{key}$ is the key of a node v regarded as a random variable. Then, the center estimation *mid* for this distribution is

$$\textit{mid}_{\textit{uniform}}(k_1, k_2) := \frac{k_1 + k_2}{2}.$$

On the topology constructed based on the keys generated by the above method, every node issued 100 search queries whose target keys $k_{\textit{target}} \in \{0, 1, \ldots, 2^{30}-1\}$ are generated by uniform distribution. We plotted the average of all path

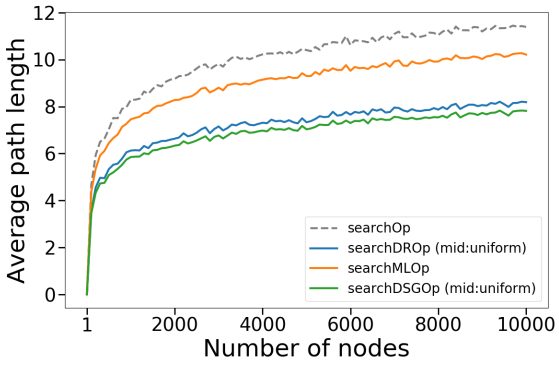

Fig. 6: Average path lengths on a topology whose keys were generated by uniform distribution.

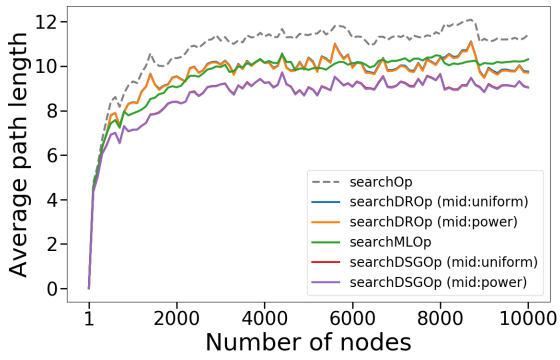

Fig. 7: Average path lengths on a topology whose keys were generated by power-law distribution.

lengths of the routings for these queries in Figure 6. The horizontal axis represents the number of participating nodes in increments of 100, and the vertical axis represents the average path length. Each line corresponds to each routing method, where `searchDSGOp(mid:uniform)` and `searchDROp(mid:uniform)` represent routings that involve the use of $mid_{uniform}$ as a center estimation for the keys. Table I is a correspondence table between names of routing algorithms and the section numbers with their descriptions. Further, every routing method is executed on the same topology for each number of nodes; and each topology is built by adding nodes to the existing topology, rather than rebuilt from scratch each time. These conditions are the same for the other experiments discussed in the subsequent sections.

As a result, the average path lengths are shorter in the order of `searchDSGOp(mid:uniform)`, `searchDROp(mid:uniform)`, `searchMLOp`, and `searchOp`. Furthermore, Detouring Skip Graph executing `searchDSGOp(mid:uniform)` shortens the average path lengths by about 32% compared to Skip Graph executing `searchOp`.

B. Generated by Power-Law Distribution

We converted pseudorandom numbers to generate keys so that the keys of participating nodes follow power-law distribution $P\{v.key \leq k\} = \int_0^k f(k)dk\ (0 \leq k \leq 2^{30})$ where $v.key$ is the key of a node v regarded as a random variable, f denotes a probability density function $f(k) = ck^{10}\ (0 \leq k \leq 2^{30})$, and c denotes a constant that satisfies $\int_0^{2^{30}} f(k)dk = 1$. Then, the center estimation *mid* for this distribution is

$$mid_{power}(k_1, k_2) := \left(\frac{k_1^{10+1} + k_2^{10+1}}{2}\right)^{\frac{1}{10+1}}.$$

The purpose of using power-law distribution is to evaluate the effect of the proposed method on biased key distribution.

On the topology constructed based on the keys generated by the above method, every node issued 100 search queries whose target keys $k_{target} \in \{0, 1, \ldots, 2^{30} - 1\}$ were generated by uniform distribution. We plotted the average of all path lengths of the routings for these queries in Figure 7. `searchDSGOp(mid:power)` and `searchDROp(mid:power)` represent routings that involve the use of mid_{power} as a center estimation of keys.

TABLE II: Average path lengths on a topology whose keys were generated by power-law distribution where $n = 100, 1000, 10000$.

	$n = 100$	1000	10000
`searchOp`	4.75	9.31	11.41
`searchDROp (mid:uniform)`	4.46	8.38	9.76
`searchDROp (mid:power)`	4.44	8.37	9.70
`searchMLOp`	4.59	7.87	10.31
`searchDSGOp(mid:uniform)`	4.35	7.14	9.05
`searchDSGOp(mid:power)`	4.34	7.15	9.03

As a result, the average path lengths were almost the same in `searchDSGOp(mid:uniform)` and `searchDSGOp(mid:power)`, and we discovered that using $mid_{uniform}$ as a detour criterion is effective even if the key distribution is biased. Table II lists the average path lengths where the number of nodes n is 100, 1000, and 10000. The numerical values also indicate that the average path lengths of the routings following `searchDSGOp(mid:uniform)` and `searchDSGOp(mid:power)` are almost the same. In both routings, the average path length of `searchOp` was shortened by about 20%.

C. Random English Titles on Wikipedia

We used random English titles obtained on Nov. 8, 2018 from API[2] published by Wikipedia as keys. Specifically, each title was considered as a single-byte character string, and we used the string as a $256 (= 2^8)$-based integer key. The purpose of using random titles is to evaluate the effect of the proposed method in realistic situations.

[2]https://www.mediawiki.org/wiki/API:Main_page (accessed Jan. 24, 2019)

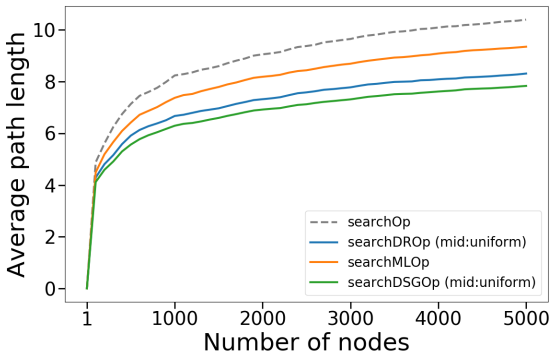

Fig. 8: Average path lengths on a topology whose keys are random English titles on Wikipedia.

On the topology constructed based on the keys obtained by the above method, every node issued search queries whose target keys are the keys of all nodes. We plotted the average of all path lengths of the routings for these queries in Figure 8.

As a result, Detouring Skip Graph executing `searchDSGOp(mid:uniform)` shortens the average path lengths by about 24% compared to Skip Graph executing `searchOp`. Although the detour criterion $mid_{uniform}$ is not a center estimation for this key distribution, it was found to be effective in shortening the path lengths.

V. Conclusion

In this paper, we proposed Detouring Skip Graph, which shortens the path lengths by using effectively the topology that Skip Graph constructs. It introduces two techniques in the routing algorithm of Skip Graph: each node 1) utilizes detour routes and 2) traverses adjacent nodes from its maximum level. In addition, we proved the reachability to the target node for any search query.

Detouring Skip Graph does not require construction of extra links and modification of its topology; thereby, it is an extension that maintains the advantages of Skip Graph. Through the evaluation experiments conducted, we confirmed that the average path length was shortened by approximately 20% to 30% compared with the value obtained using Skip Graph. Further, it was experimentally found that $mid_{uniform}$, the average of the keys belonging to two nodes, is effective as a detour criterion even for biased or realistic key distribution.

In the future, we plan to address analytical evaluation of shortening path lengths and application of the proposed algorithm to other extensions of Skip Graph.

References

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," in *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220.

[2] N. Ramzan, H. Park, and E. Izquierdo, "Video streaming over P2P networks: Challenges and opportunities," *Signal Processing: Image Communication*, vol. 27, no. 5, pp. 401–411, May 2012.

[3] A. Yahyavi and B. Kemme, "Peer-to-peer Architectures for Massively Multiplayer Online Games: A Survey," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 9:1–9:51, Jul. 2013.

[4] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an Optimized BlockChain for IoT," in *Proceedings of the Second IEEE/ACM International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI '17. New York, NY, USA: ACM, 2017, pp. 173–178.

[5] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "LightChain: A DHT-based Blockchain for Resource Constrained Environments," *arXiv:1904.00375 [cs]*, Mar. 2019.

[6] J. Aspnes and G. Shah, "Skip graphs," *ACM Transactions on Algorithms*, vol. 3, no. 4, pp. 37:1–37:25, Nov. 2007.

[7] S. Huq and S. Ghosh, "Locally Self-Adjusting Skip Graphs," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2017, pp. 805–815.

[8] A. Goyal, S. Batra, N. Kumar, G. S. Aujla, and M. S. Obaidat, "Adaptive Skip Graph Framework for Peer-to-Peer Networks: Search Time Complexity Analysis," in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–6.

[9] W. Pugh, "Skip lists: A Probabilistic Alternative to Balanced Trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, Jun. 1990.

[10] K. Higuchi, M. Yoshida, T. Tsuji, and N. Miyamoto, "Correctness of the routing algorithm for distributed key-value store based on order preserving linear hashing and skip graph," in *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Jun. 2017, pp. 459–464.

[11] M. Naor and U. Wieder, "Know Thy Neighbor's Neighbor: Better Routing for Skip-Graphs and Small Worlds," in *Peer-to-Peer Systems III*. Springer, Berlin, Heidelberg, Feb. 2004, pp. 269–277.

[12] A. G. Beltran, P. Sage, and P. Milligan, "Skip Tree Graph: A Distributed and Balanced Search Tree for Peer-to-Peer Networks," in *2007 IEEE International Conference on Communications*, Jun. 2007, pp. 1881–1886.

[13] F. Makikawa, T. Tsuchiya, and T. Kikuno, "Balance and Proximity-Aware Skip Graph Construction," in *2010 First International Conference on Networking and Computing*, Nov. 2010, pp. 268–271.

[14] T. Kawaguchi, R. Banno, M. Hojo, M. Ohnishi, and K. Shudo, "Self-Refining Skip Graph: Skip Graph Approaching to an Ideal Topology," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan. 2017, pp. 441–448.

[15] P. Keef and D. Guichard, "Introduction to Higher Mathematics," https://www.whitman.edu/mathematics/higher_math_online/.