一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SITE2018-86,IA2018-78 (2019-03)

Detouring Skip Graph: 迂回経路を活用する構造化オーバレイ

金子　孟司† 　坂野　遼平†† 　青木　優介† 　首藤　一幸†††

†, ††, ††† 東京工業大学
〒 152-8550 東京都目黒区大岡山 2-12-1
E-mail: †{kaneko.t.ay,aoki.y.au}@m.titech.ac.jp, ††banno@computer.org, †††shudo@is.titech.ac.jp

あらまし　ノード群による自律分散的なネットワークを構築し，効率的なルーティングを実現する構造化オーバレイ技術の一つに Skip Graph がある．Skip Graph は，各ノードに割り当てられる membership vector に基づいてネットワークトポロジを形成することで，ノード数 n に対して $O(\log n)$ の経路長を達成する．しかし，各ノードは大域的な情報をもたないため，大抵の場合は最短経路ではない．そこで我々は，迂回経路を活用することで経路長を短縮する Detouring Skip Graph を提案する．提案手法は追加リンクの構築等を必要としないため，Skip Graph の特長を維持した上での経路長の短縮に成功している．評価実験により，平均経路長が Skip Graph よりも 20% から 30% 程度短縮したことを確認した．

キーワード　Skip Graph，構造化オーバレイ，ルーティングアルゴリズム，迂回経路

Detouring Skip Graph: A Structured Overlay Utilizing Detour Routes

Takeshi KANEKO†, Ryohei BANNO††, Yusuke AOKI†, and Kazuyuki SHUDO†††

†, ††, ††† Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8552 Japan
E-mail: †{kaneko.t.ay,aoki.y.au}@m.titech.ac.jp, ††banno@computer.org, †††shudo@is.titech.ac.jp

Abstract　Skip Graph, one of the structured overlays, constructs its own structure based on membership vectors assigned to every node, and consequently it provides routing path lengths of $O(\log n)$ where n is the total number of nodes. However, there is a problem that most of routing paths are not the shortest paths because each node knows only its local information, rather than the global topology. We proposed Detouring Skip Graph, which shortens the path lengths by means of utilizing detour routes. It does not require construction of extra links and modification of its topology; thereby, we succeeded in shortening them while maintaining the advantages of Skip Graph. Our evaluation experiments confirmed that the average path length was shortened by approximately 20% to 30% compared to Skip Graph.

Key words　Skip Graph, Structured overlays, Routing algorithms, Detour routes

1. はじめに

オーバレイネットワークとは，インターネット等の既存ネットワーク上に構築されるアプリケーションレベルの論理ネットワークである．特に構造化オーバレイは，特定のデータ構造やプロトコルに従って自律分散的なネットワークを構築することで，目的ノードへの到達性保証や高いスケーラビリティ，効率的なルーティング等を提供する．

分散ハッシュテーブル（Distributed Hash Table，DHT）の機能を有する構造化オーバレイ [1] [2] は，ハッシュ値をキーとしてそのキーと値のペアでデータを分散管理する．ハッシングによる一様なキー空間は，偏りのないトポロジを形成して負荷分散や経路長の平均化に寄与する．しかし，キーの順序関係を保存しないハッシングは，ある範囲内に含まれるすべてのキーを検索するクエリといった複雑な検索クエリを実現困難にする．

一方で，範囲検索が可能な構造化オーバレイの一つに Skip Graph [3] がある．これはハッシングをせずにデータを管理するため，キーの順序を保存した状態でトポロジを形成する．ノード数を n とすると，Skip Graph はショートリンクを適切に使用したルーティングによって $O(\log n)$ の経路長を達成する．しかし，各ノードは大域的な情報をもたないため，構築し

This is an unrefereed paper.

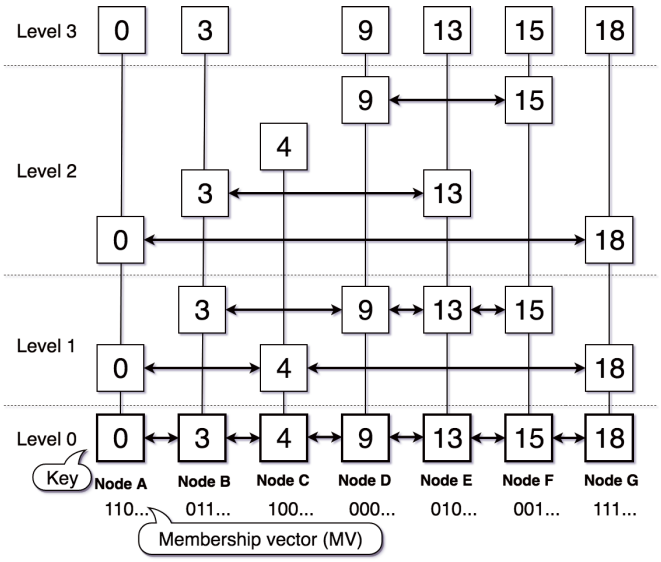


図 1: Skip Graph の例

Fig. 1 An example of Skip Graph.

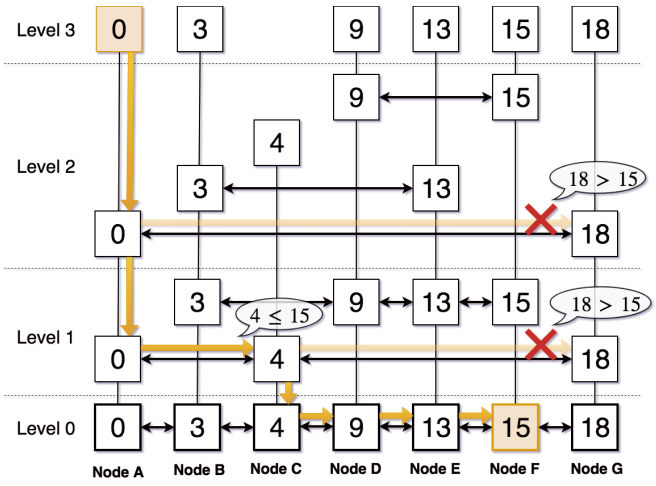


図 2: キー 15 の検索クエリに対する searchOp によるルーティング

Fig. 2 A searchOp routing from node A to search key 15.

たリンクを最大限に使用することができず，最短経路よりも大きく劣る経路を選択する傾向にある．

そこで本稿では，構築済みのリンクをより効果的に使用して経路長を短縮する Detouring Skip Graph を提案する．提案手法は追加リンクの構築等を必要としないため，Skip Graph の特長を維持した上での経路長の短縮に成功している．具体的には Skip Graph のルーティングアルゴリズムに対し，1) 最大レベルからの探索 と 2) 迂回経路の活用 の 2 つの変更を加える．

本稿の構成は以下の通りである．2. で Skip Graph の概要とその経路長短縮に関連する研究について述べる．次に 3. で Detouring Skip Graph の詳細を述べ，4. で提案手法の評価実験の内容とその結果を述べる．最後に 5. で本研究のまとめと今後の課題について述べる．

2. 関 連 研 究

2.1 Skip Graph

Skip Graph は Skip List [4] を元に設計され，そのトポロジは各ノードが複数のソートされた双方向連結リストに属した構造になっている．図 1 は Skip Graph が形成するトポロジの例である．各ノードは全順序な性質をもつキーと membership vector (MV) と呼ばれるランダムな文字列をもち，それらの値に基づいてトポロジを決定する．図中では MV の要素であるアルファベット全体の集合を $\{0,1\}$ としている．レベル i にある連結リストは，MV の先頭 i 桁が一致するノード同士で双方向に接続する．特にレベル 0 では全ノードが一つの連結リストに属する．よって，Skip Graph のノードの個数を n とすると，各ノードの属する連結リストの個数は $O(\log n)$ となり，このトポロジ上で Skip List と同様なルーティング（探索）を実行することによって，あるキー k_{target} を検索するようなクエリは $O(\log n)$ の経路長を達成する．詳細なルーティングアルゴリズムは次のとおりである．ただし，連結リスト内のキー順序は左から右の方向へ昇順であるものとする．

Skip Graph に属するあるノード v_{current} が検索クエリ searchOp を受信したとする．このとき，このクエリは開始ノード v_{start} と検索キー k_{target} と受信時のレベル l_{prev} の情報を伴う．v_{current} のキー v_{current}.key が k_{target} と一致する場合は，検索に成功したため v_{start} へ発見クエリ foundOp を送信する．v_{current}.key $< k_{\text{target}}$ の場合は，l_{prev} からレベルを降順に辿って右方の隣接ノードを探索し，k_{target} 以下のキーをもつ最初のノードを送信先に決定して検索クエリ searchOp を送信する．v_{current}.key $> k_{\text{target}}$ の場合も同様の方法で左方の送信先ノードを決定して検索クエリ searchOp を送信する．

図 2 は図 1 のトポロジ上で，ノード A でキー $k_{\text{target}} = 15$ の検索クエリが発行され，各ノードが上述の処理を実行したときのルーティングを表す．経由するノードのキー列が $(0, 4, 9, 13, 15)$ となり，経路長は 4 である．

2.2 経路長の短縮

前述の例では経路長が 4 の経路となったが，実際にはより短い経路が存在する．例えばノード A がレベル 1 の隣接ノード G を送信先ノードに選択すると，経由するノードのキー列は $(0, 18, 15)$ となり，このクエリに対する経路長は 2 に短縮する（このときのノード A からノード G への経路のような，ノードがもつキーと検索キー k_{target} の大小関係が前後で逆転する経路を以降では迂回経路と呼ぶことにする）．このように Skip Graph のルーティングは構築済みのリンクを十分に活用できていない点で非効率である．そこで，ルーティングあるいはトポロジを拡張することで経路長を短縮する様々なアプローチが提案されている．

Higuchi らは，線形ハッシュ法を使用してデータを分割することにより偏りのない Skip Graph のトポロジを構築する手法を提案している [5]．そのトポロジ上で，各ノードは経路長を短縮する迂回経路を検出してルーティング時にその迂回経路に経路変更する．これによって結果的に経路長を短縮することが可能である．迂回経路を用いる点で後述の提案手法と

類似しているが，これは一般の Skip Graph を対象にはしていない．

既存の経路長を短縮する手法としては，Skip Graph に追加リンクを構築してそのトポロジ上の適切なルーティングを実行する手法 [6] [7] や，Skip Graph のトポロジを再構築・修正することで乱数に依存した非均一なトポロジを理想的なトポロジに近づける手法 [8] [9] が存在する．しかし，追加リンクの構築やトポロジの修正は，必要な転送メッセージの増加や転送方法の複雑化を招く．これは，トポロジの安定に要する時間の長期化や churn 耐性の低下，下位ネットワークの通信帯域の圧迫等が危惧される．

3. 提 案 手 法

以降ではキー集合を K とおき，K は有理数体の部分集合であると仮定する(注1)．つまり，キーに対して四則演算や絶対値等が定義されているものとする．また，同一のキーを複数のノードがもたないことを仮定する．

Detouring Skip Graph は，Skip Graph のルーティングアルゴリズムを改良することによって経路長を短縮する．変更点は 1) 最大レベルからの探索 と 2) 迂回経路の活用 の 2 つである．これらは独立した変更であるため，まず 3.1 と 3.2 でそれぞれの変更後のアルゴリズムを述べ，続いて 3.3 でこれらを組み合わせた Detouring Skip Graph の詳細を述べる．

3.1 最大レベルからの探索

Skip Graph のルーティングでは 2.1 で述べたように，ノード v_{current} は受信時のレベル l_{prev} から降順に隣接ノードを探索し，条件を満たすノードを発見するとそのノードを送信先に決定する．ルーティング全体をみるとレベルが単調減少であることがわかる．しかし l_{prev} より大きいレベルの隣接ノードが条件を満たす場合もあり，またレベルが大きいほど隣接ノード間のキーの大小差は大きい．このことから，各ノードがレベル l_{prev} から探索するよりも v_{current} の最大レベルから探索する方が，送信先ノードのキーと検索キー k_{target} の差が小さい（または等しい）ことがわかる．ここでノード v の最大レベル v.maxLevel とは，v がもつリンクのレベルの最大値を意味する．つまり，毎クエリで最大レベルから探索することは最終的な経路長を短縮するはたらきがある．

Algorithm 1 はこのアルゴリズムの疑似コードである．あるノード v_{current} は検索クエリ `searchMLOp` を受信すると，v_{current} の最大レベル v_{current}.maxLevel から隣接ノードの探索を開始し，条件を満たすノードを発見するとそのノードへクエリ `searchMLOp` を送信する．Skip Graph のルーティングアルゴリズムとの相違点は，探索開始時のレベルのみである．なお，レベルの探索を線形探索ではなく二分探索にすることも可能でありその方が高速ではあるが，疑似コード中では単純さを優先して線形探索を実行している．

図 3 は図 1 のトポロジ上で，ノード A でキー $k_{\text{target}} = 15$

(注1)：一般に可算全順序集合は有理数体に順序集合として埋め込むことが可能である [10].

Algorithm 1 : `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; return
4:   else if v_current.key < k_target then
5:     for l_current ← v_current.maxLevel downTo 0 do
6:       v_next ← v_current.neighbors[R][l_current]
7:       if v_next.key ≤ k_target then
8:         send ⟨searchMLOp, v_start, k_target⟩ to v_next
9:         return
10:      end if
11:    end for
12:  else
13:    for l_current ← v_current.maxLevel downTo 0 do
14:      v_next ← v_current.neighbors[L][l_current]
15:      if v_next.key ≥ k_target then
16:        send ⟨searchMLOp, v_start, k_target⟩ to v_next
17:        return
18:      end if
19:    end for
20:  end if
21:  send ⟨notFoundOp, v_current⟩ to v_start
22: end upon
```

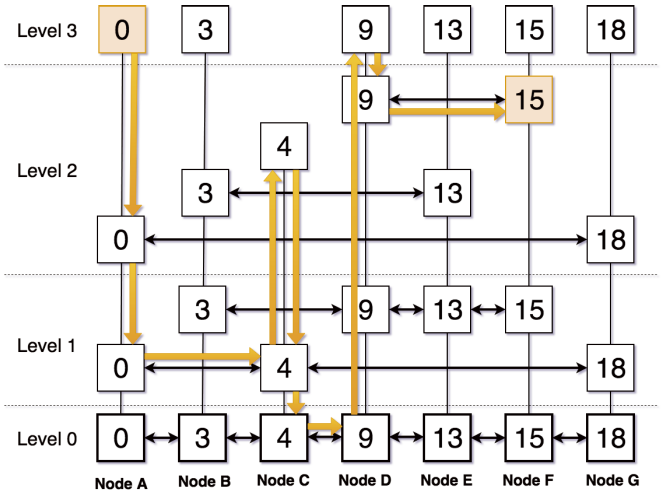


図 3: キー 15 の検索クエリに対する `searchMLOp` によるルーティング

Fig. 3 A `searchMLOp` routing from node A to search key 15.

の検索クエリが発行され，各ノードが Algorithm 1 を実行したときのルーティングを表す．経由するノードのキー列が $(0, 4, 9, 15)$ となり，経路長は 3 である．これは図 2 の経路長 4 より短い．

Algorithm 1 に経路長を短縮する利点がある一方で，クエリを受信してから送信先ノードを決定するまでの計算コストは逆に大きくなるという欠点は留意したい．全ノードの最大レベルの最大値を $l_{\text{MAX}}(= O(\log n))$ とし，経路長を $H(= O(\log n))$ とおくと，ルーティング完了までに各ノードが経路選択に要する（通信時間や I/O の処理時間を含めない）時間の合計

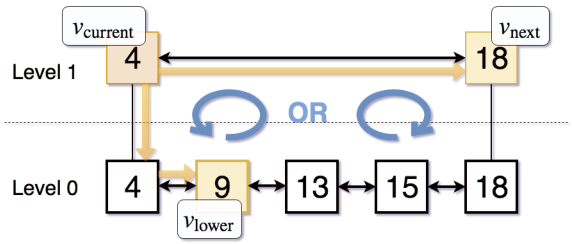


図 4: ノード v_{current} は迂回経路を選択するべきか
Fig. 4 Should node v_{current} select the detour route?

は，Skip Graph が $O(l_{\text{MAX}} + H) = O(\log n)$ であるのに対し，Algorithm 1 は $O(H \cdot l_{\text{MAX}}) = O(\log^2 n)$，特に二分探索を実行する場合は $O(H \log l_{\text{MAX}}) = O(\log n \cdot \log(\log n))$ であり，計算量で Skip Graph に劣っている．しかし，ルーティングにかかる時間は通信にかかる時間が支配的であることが大半であり，経路長を短縮することの方が多くの場合では重要である．

3.2 迂回経路の活用

図 4 は図 1 のトポロジの一部分である．ノード v_{current} のレベル 1，0 での右の隣接ノードをそれぞれ v_{next}，v_{lower} とする．今，キー $k_{\text{target}} (9 \leq k_{\text{target}} < 18)$ を検索するクエリをノード v_{current} が受信したとする．このとき，Skip Graph のルーティングに従う場合は次のノードとして v_{lower} が選択される．$k_{\text{target}} = 15$ ならば経路上のキー列は $(4, 9, 13, 15)$ で経路長は 3 となるが，迂回経路として次ノードに v_{next} を選択する場合を考えると経路上のキー列は $(4, 18, 15)$ で経路長は 2 となり，後者の方が経路長が短い．逆に $k_{\text{target}} = 13$ ならば v_{lower} を選択した方が経路長が短い．

v_{next} と v_{lower} のどちらを選択した方が短い経路になるのかは，図 4 からわかるように下位レベル（図ではレベル 0）のノード列の中央が基準となる．本節では，この考察を各レベルに対し拡張して迂回経路の選択基準にしたルーティングアルゴリズムを述べる．Algorithm 2 はその疑似コードである．

クエリ `searchDROp` を受信したノードは，Skip Graph のルーティングと同様にレベル l_{prev} から降順に隣接ノードを探索するが，探索の途中で迂回した方が良いと判断した場合はその迂回先のノードを送信先に選択する．迂回判定には関数 $\text{closeToRight}(k_{\text{target}}, k_{\text{left}}, k_{\text{right}})$ を使用する．これは，k_{target} の k_{right} との差が $\text{mid}(k_{\text{left}}, k_{\text{right}})$ の k_{right} との差よりも小さいときに true を，それ以外には false を返す．mid はネットワークの構築前に定める関数であり，キーの分布を考慮して次を満たすように設定する：

任意時点での参加ノード全体の集合 V と $^{\forall}k_1, k_2 \in K (k_1 \leq k_2)$ に対して（1）と（2）が概ね等しい．

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

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

例えば参加ノードのキー $v.\text{key}$ を確率変数とみなしたときに，$K = \{0, 1, \cdots, n-1\}$ かつ $P\{v.\text{key} = k\} = \frac{1}{n}$ ならば $\text{mid}(k_1, k_2) := \frac{k_1+k_2}{2}$，$K = \{0, 1, \cdots\}$ かつ $P\{v.\text{key} = k\} = \left(\frac{1}{2}\right)^{k+1}$ ならば $\text{mid}(k_1, k_2) := -\log_2 \left(\frac{2\left(\frac{1}{2}\right)^{k_1} + \left(\frac{1}{2}\right)^{k_2}}{3} \right)$

Algorithm 2 : `searchDROp` in node v_{current}

▷ using Detour Routes

```
1: upon receiving ⟨searchDROp, v_start, k_target, l_prev⟩ then
2:   if v_current.key = k_target then
3:     send ⟨foundOp, v_current⟩ to v_start; return
4:   else if v_current.key < k_target then
5:     for l_current ← l_prev downTo 0 do
6:       v_next ← v_current.neighbors[R][l_current]
7:       if v_next.key ≤ k_target then
8:         send ⟨searchDROp, v_start, k_target, l_current⟩ to v_next
9:         return
10:      else if l_current > 0 then
11:        v_lower ← v_current.neighbors[R][l_current − 1]
12:        if closeToRight(k_target, v_lower.key, v_next.key) then
13:          send ⟨searchDROp, v_start, k_target, l_current⟩ to v_next
14:          return
15:        end if
16:      end if
17:    end for
18:  else
19:    for l_current ← l_prev downTo 0 do
20:      v_next ← v_current.neighbors[L][l_current]
21:      if v_next.key ≥ k_target then
22:        send ⟨searchDROp, v_start, k_target, l_current⟩ to v_next
23:        return
24:      else if l_current > 0 then
25:        v_lower ← v_current.neighbors[L][l_current − 1]
26:        if not closeToRight(k_target, v_next.key, v_lower.key)
   then
27:          send ⟨searchDROp, v_start, k_target, l_current⟩ to v_next
28:          return
29:        end if
30:      end if
31:    end for
32:  end if
33:  send ⟨notFoundOp, v_current⟩ to v_start
34: end upon
35: function closeToRight(k_target, k_left, k_right)
36:   k_mid ← mid(k_left, k_right)
37:   return k_mid < k_target
38: end function
```

である．このように mid を定めることで `searchDROp` の各ステップにおいて，$\text{mid}(v_{\text{lower}}.key, v_{\text{next}}.key)$ または $\text{mid}(v_{\text{next}}.key, v_{\text{prev}}.key)$ が下位レベルのノード列の中央を推定したキーとなり，関数 closeToRight が適切な迂回判定を果たしていることがわかる．

実際にはキーの分布をあらかじめ得ることはむずかしいが，4. の評価実験によって多くの場合では

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

と設定することで十分に経路長を短縮することが可能であるとわかった．

図 5 は図 1 のトポロジ上で，ノード A でキー $k_{\text{target}} = 15$ の

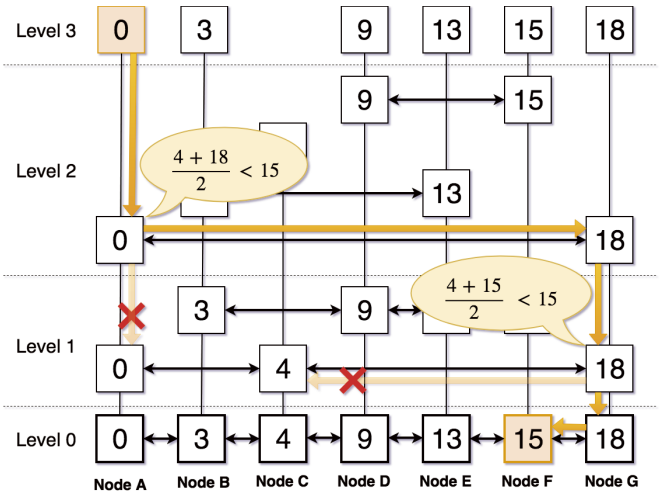


図 5: キー 15 の検索クエリに対する searchDROp によるルーティング

Fig. 5 A searchDROp routing from node A to search key 15 where $\mathrm{mid}(k_1, k_2) = \frac{k_1+k_2}{2}$.

検索クエリが発行され，各ノードが Algorithm 2 を実行したときのルーティングを表す．ただし $\mathrm{mid}(k_1, k_2) := \frac{k_1+k_2}{2}$ としている．経由するノードのキー列が $(0, 18, 15)$ となり，経路長は 2 である．これは図 2 の経路長 4 より短い．

3.3 Detouring Skip Graph

以上で述べた 2 つのルーティングの改良アルゴリズムは独立したロジックなため，これらを融合したアルゴリズムは自然に構成できる．そこでそのようなルーティングを実行する Skip Graph を Detouring Skip Graph と命名することとする．

Detouring Skip Graph に参加する各ノード v_{current} は検索クエリ searchDSGOp を受信すると，3.1 で述べたように v_{current} の最大レベルから隣接ノードの探索を開始し，条件を満たすノードを発見するとそのノードへクエリ searchDSGOp を送信する．その過程において 3.2 で述べたように迂回判定を行って適切に迂回経路を選択する．

これによって Detouring Skip Graph は検索クエリの経路長短縮を実現する．これは 2.2 で述べた既存手法とは異なり，追加リンクの構築やトポロジの修正をしないため，転送メッセージや管理コストの増加は一切必要としない．また，Skip Graph が備える優れた性質も維持した拡張になっている．

ところで Detouring Skip Graph には，ルーティングを実行したときの経路上のノードのキー列が順序に関して単調でないという特徴がある．これはキー列が単調である Skip Graph とは異なる点であり，ルーティング時の経路が無限ループする心配がある．しかし，3.3.1 で述べるように到達性は保証される．

3.3.1 到達性の証明

本節では Detouring Skip Graph の検索クエリに対する目的ノードへの到達性を証明する．ただし，Detouring Skip Graph がネットワークとして分離しないことを前提としている．

今，Detouring Skip Graph 上でキー k を検索するクエリが発行されたとする．このとき各変数を以下のように定義する．

［定義 1］

- $(a_1, a_2, \cdots) :=$ 経路上のノードのキー列．
- $S_k := \{x \in K \mid x < k\}$.
- $T_k := \{x \in K \mid x > k\}$.
- $a_i \in S_k$ を満たす a_i をすべて集めた $(a_i)_i$ の部分列を $(a_{s_1}, a_{s_2}, \cdots)$ とする．
- $a_i \in T_k$ を満たす a_i をすべて集めた $(a_i)_i$ の部分列を $(a_{t_1}, a_{t_2}, \cdots)$ とする．
- $S_k \times T_k$ 上の二項関係 $\succ_k$，$\prec_k$ を
  - $\succ_k := \{(x, y) \in S_k \times T_k \mid \mathrm{closeToRight}(k, x, y)\}$
  - $\prec_k := \{(x, y) \in S_k \times T_k \mid \neg\mathrm{closeToRight}(k, x, y)\}$

  で定める．

このとき，キー列 $(a_i)_i$ が有限列であることとクエリが目的ノードへ到達することは同値であるため，$(a_i)_i$ が有限列であることを示せば良い．ここでキー列 $(a_{s_j})_j, (a_{t_j})_j$ が k に真に近づく，つまりそれぞれが狭義単調増加，狭義単調減少であることを示せば，参加ノードが有限であることから $(a_i)_i$ が目的ノードのキーへ有限回で収束し，$(a_i)_i$ が有限列であることが導かれる．

次に各部分列の狭義単調性を示すために，迂回判定で使用する関数 mid に次の性質を要求する．

［性質 1］ $\forall x \in S_k, \forall y \in T_k$ に対して次を満たす．

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

$\mathrm{mid}(x, y) = \frac{x+y}{2}$ や，任意の確率分布に基づいて推定される中央値で関数 mid を定義したとき，mid は［性質 1］を満たす．

このとき次の補題が成り立つ．

［補題 1］ $\forall x_1, x_2 \in S_k, \forall y_1, y_2 \in T_k$ に対して次を満たす．

- $\left[\exists x \in S_k\,(x \prec_k y_1 \wedge x \succ_k y_2)\right] \Rightarrow y_1 < y_2$
- $\left[\exists y \in T_k\,(x_1 \succ_k y \wedge x_2 \prec_k y)\right] \Rightarrow x_1 > x_2$

証明 $x \prec_k y_1$ かつ $x \succ_k y_2$ を満たす $x \in S_k$ が存在するとする．$y_1 \geq y_2$ ならば $x \succ_k y_2$ と［性質 1］より $x \succ_k y_1$ となるが $x \prec_k y_1$ と矛盾する．よって $y_1 < y_2$ となり前者の命題は成り立つ．後者の命題も同様にして成り立つ．□

この補題によって次の定理が成り立つ．

［定理 1］ キー列 $(a_{s_j})_j, (a_{t_j})_j$ はそれぞれ狭義単調増加，狭義単調減少である．

証明 各ステップ i に対して

$$\begin{cases} s_j = i \text{ となる } j \text{ が存在するならば } & \left[j > 1 \Rightarrow a_{s_j} > a_{s_{j-1}}\right] \\ t_j = i \text{ となる } j \text{ が存在するならば } & \left[j > 1 \Rightarrow a_{t_j} < a_{t_{j-1}}\right] \end{cases} \quad (*)$$

が成り立つことを示せば十分である．ここでステップ i とはルーティングの経路上の i 番目のノードでの処理を表す．これを帰納法で示す．

今，ステップ $1,2,\cdots,i$ で（$*$）が成り立つと仮定する．（I）$a_i = k$ ならば，ルーティングが終了するためステップ $i+1$ は存在しない．（II）$a_i \in S_k$ ならば（i）$a_{i+1} \in S_k$ の場合と（ii）$a_{i+1} \in T_k$ の場合と（iii）$a_{i+1} = k$ の場合と（iv）a_{i+1} が存在しない場合がある．（i）の場合は $a_i < a_{i+1}$ より（iii）の場合は $a_{i+1} \notin S_k \cup T_k$ よりステップ $i+1$ で（$*$）が成り立つ．（iv）の場合はルーティングが終了するためステップ $i+1$ は存在しない．（ii）の場合は迂回経路を選択したことを意味するので，$t_j = i+1$ となる j と $x > a_i$ となる $x \in S_k$ が存在して $x \prec_k a_{t_j}$ であり，［性質 1］より $a_i \prec_k a_{t_j}$ である．$j > 1$ ならば部分列 $(a_{t_{j'}})_{j'}$ の構成方法よりステップ t_{j-1} で迂回経路を選択し，ステップ $t_{j-1}+1, t_{j-1}+2, \cdots, i-1$ で迂回経路を選択していないことがわかる．よって $a_{t_{j-1}+1}, a_{t_{j-1}+2}, \cdots, a_i \in S_k$ かつ $y < a_{t_{j-1}}$ となる $y \in T_k$ が存在して $a_{t_{j-1}+1} \succ_k y$ が成り立ち，［性質 1］より $a_{t_{j-1}+1} \succ_k a_{t_{j-1}}$ となる．また，帰納法の仮定より $a_{t_{j-1}+1} < a_{t_{j-1}+2} < \cdots < a_i$ であり，再び［性質 1］より $a_i \succ_k a_{t_{j-1}}$ である．よって［補題 1］より $a_{t_j} < a_{t_{j-1}}$ が成り立ち，ステップ $i+1$ で（$*$）が成り立つ．（III）$a_i \in S_k$ ならば（II）と同様にしてステップ $i+1$ で（$*$）が成り立つ．□

以上より，任意の単一検索クエリに対して目的ノードへの到達性が保証された．

4. 評　　価

シミュレーションによって経路長の評価実験を行い，提案手法が経路長に与える影響を観察した．キーの生成方法として次の 3 通りを試した．

- 一様分布で生成した乱数
- べき分布で生成した乱数
- ランダムな英字タイトル

各実験の詳細は以降の各節で述べる．なお，membership vector のアルファベット集合は $\{0,1\}$ で固定している．

4.1 一様分布で生成した乱数

参加ノードのキーを確率変数 v.key とみなして $P\{v.\text{key} = k\} = \frac{1}{2^{30}}$ $(k \in \{0,1,\cdots,2^{30}-1\})$ の一様分布に従うように，疑似乱数を使用してキーを生成した．このキー分布における中央の推定値 mid は

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

である．

図 6(a) には，上述の方法で生成したキーを元に構築したトポロジ上で，各ノードからそれぞれ一様にランダムな 100 個のキー $k_{\text{target}} \in \{0,1,\cdots,2^{30}-1\}$ の検索クエリを発行したときの，全経路長の平均値をグラフにプロットした．横軸が参加ノード数で縦軸が平均経路長を表し，各線は各ルーティング手法に対応している．また，トポロジは各ルーティングで同一のものを扱い，参加ノード数は 100 刻み，トポロジは参加ノード数が変化するたびに毎回再構築するのではなく，すでにあるトポロジにノードを追加する方法で構築している．これらは後述する他のキー分布による実験でも同様である．次にノード図 6(b) には参加ノード数 $n = 10000$ で固定して，ある経路長となる経路が何本存在するかの分布をプロットした．同じく各ルーティングで同一のトポロジを使用している．ここで `searchDSGOp(mid:uniform)`，`searchDROp(mid:uniform)` は迂回基準に関数 $\text{mid}_{\text{uniform}}$ を使用したルーティングを表す．表 1 は各ルーティングのアルゴリズム名とその記述がある節番号の対応表である．

結果として平均経路長は `searchDSGOp(mid:uniform)`，`searchDROp(mid:uniform)`，`searchMLOp`，`searchOp` の順に短く，`searchDSGOp(mid:uniform)` を実行する Detouring Skip Graph は，`searchOp` を実行する Skip Graph の平均経路長を約 32% 短縮した．

4.2 べき分布で生成した乱数

参加ノードのキーを確率変数 v.key とみなして，確率密度関数 $f(k) = ck^{10}$ $(0 \le k \le 2^{30})$ に対して $P\{v.\text{key} \le k\} = \int_0^k f(k)dk$ $(0 \le k \le 2^{30})$ のべき分布に従うように，疑似乱数の値に変換を加えることでキーを生成した．ここで c は $\int_0^{2^{30}} f(k)dk = 1$ を満たす定数である．ただしプログラムの都合上，小数点以下を切り捨てして $k \in \{0,1,\cdots,2^{30}-1\}$ としている．このキー分布における中央の推定値 mid は

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

である．べき分布を使用した目的は，偏ったキー分布に対する提案手法の効果を評価するためである．

図 7(a) には，上述の方法で生成したキーを元に構築したトポロジ上で，各ノードからそれぞれ一様にランダムな 100 個のキー $k_{\text{target}} \in \{0,1,\cdots,2^{30}-1\}$ の検索クエリを発行したときの，全経路長の平均値をグラフにプロットした．図 7(b) には参加ノード数 $n = 10000$ で固定して，ある経路長となる経路が何本存在するかの分布をプロットした．ここで `searchDROp(mid:power)`，`searchDSGOp(mid:power)` は迂回基準に関数 $\text{mid}_{\text{power}}$ を使用したルーティングを表す．

結果として平均経路長は `searchDSGOp(mid:uniform)` と `searchDSGOp(mid:power)` で同程度となり，キー分布が偏った場合でも迂回基準に $\text{mid}_{\text{uniform}}$ を使用することが効果的であることがわかった．表 2 にはノード数 $n = 100, 1000, 10000$ のときの平均経路長を載せたが，数値からも同等であると判断できる．また，両者とも `searchOp` の平均経路長を約 20% 短縮した．

4.3 ランダムな英字タイトル

Wikipedia(注2)が公開している API(注3)からランダムな英字タイトルを取得(注4)し，タイトルを 1 バイト文字の列とみなし $256(=2^8)$ 進数の整数として扱ったものをキーとして使用した．ランダムな英字タイトルを使用した目的は現実的な状況における提案手法の効果を評価するためである．

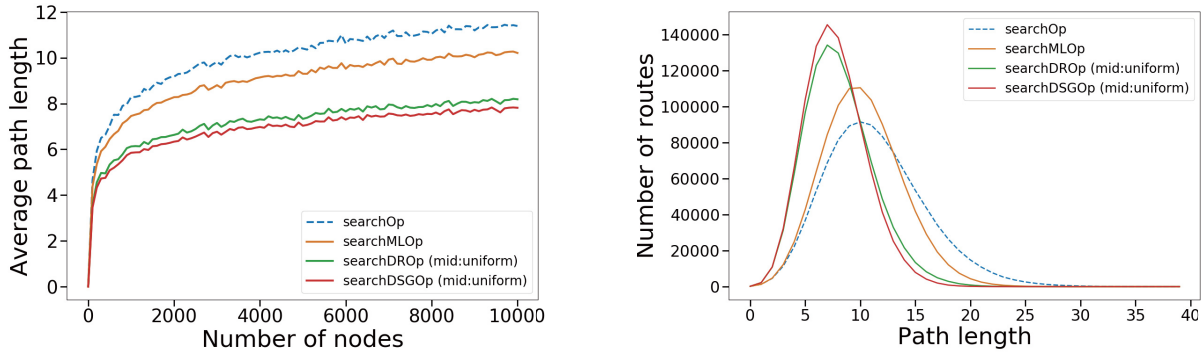
図 8(a) には，上述の方法で生成したキーを元に構築したト

(注2)： `https://www.wikipedia.org/` (accessed Jan. 24, 2019)
(注3)： `https://www.mediawiki.org/wiki/API:Main_page` (accessed Jan. 24, 2019)
(注4)： 2018 年 11 月 8 日に取得

`searchOp`	：Skip Graph における従来のルーティング手法	（Sec. 2.1）
`searchMLOp`	：最大レベルから探索するルーティング手法	（Sec. 3.1）
`searchDROp`	：迂回経路を活用するルーティング手法	（Sec. 3.2）
`searchDSGOp`	：Detouring Skip Graph におけるルーティング手法	（Sec. 3.3）

表 1: 実験対象アルゴリズムの一覧表

Table 1　A List of algorithms used as evaluation subjects.



(a) 平均経路長

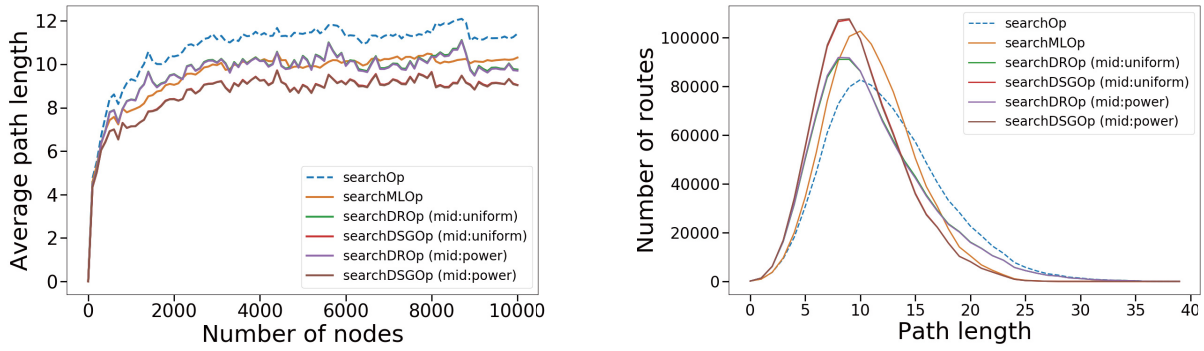
(a) Average path lengths.

(b) 経路長の分布，ノード数 $n = 10000$

(b) Distribution of path lengths where $n = 10000$.

図 6: 一様分布で生成した乱数をキーとしたトポロジ上での経路長比較

Fig. 6　Comparison of path lengths on a topology whose keys were generated by uniform distribution.



(a) 平均経路長

(a) Average path lengths.

(b) 経路長の分布，ノード数 $n = 10000$

(b) Distribution of path lengths where $n = 10000$.

図 7: べき分布で生成した乱数をキーとしたトポロジ上での経路長比較

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

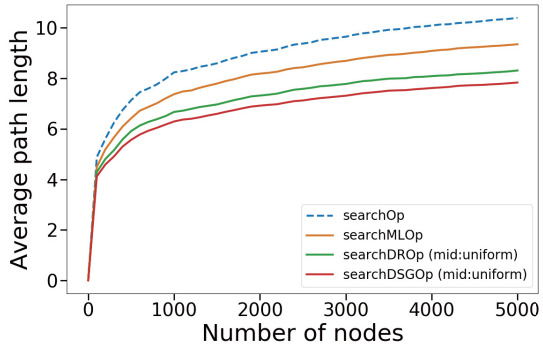
ポロジ上で，各ノードから各ノードのキーへの検索クエリを発行したときの，全経路長の平均値をグラフにプロットした．図 8(b) には参加ノード数 $n = 5000$ で固定して，ある経路長となる経路が何本存在するかの分布をプロットした．

結果として `searchDSGOp(mid:uniform)` を実行する Detouring Skip Graph は，`searchOp` を実行する Skip Graph の平均経路長を約 24% 短縮した．迂回基準 $\mathrm{mid}_{\mathrm{uniform}}$ はこのキー分布に対する中央の推定値ではないが，経路長短縮に有効であることがわかった．

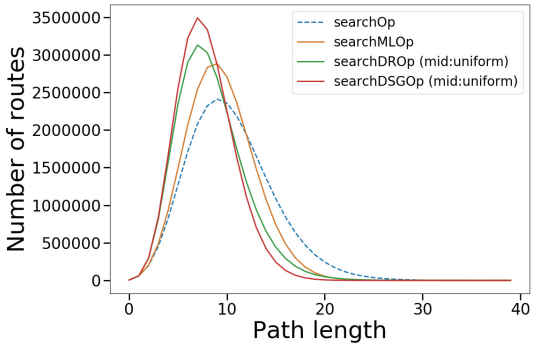
5. ま と め

本稿では，Skip Graph が形成するトポロジをより効果的に使用することによって経路長を短縮する Detouring Skip Graph を提案した．これは経路長短縮のために追加リンクの構築やトポロジの修正が一切不要であり，Skip Graph の特長を維持した拡張になっている．

Detouring Skip Graph は Skip Graph のルーティングに 1) 最大レベルからの探索 と 2) 迂回経路の活用 の 2 つの変更を加える．評価実験によって平均経路長が 20% から 30% 程度短縮したこと，さらに 2 ノードがもつキーの平均値による迂回



(a) 平均経路長
(a) Average path lengths.



(b) 経路長の分布，ノード数 $n = 5000$
(b) Distribution of path lengths where $n = 5000$.

図 8: ランダムな英字タイトルをキーとしたトポロジ上での経路長比較
Fig. 8 Comparison of path lengths on a topology whose keys are random English titles on Wikipedia.

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

表 2: べき分布で生成した乱数をキーとしたトポロジ上での平均経路長，ノード数 $n = 100, 1000, 10000$
Table 2 Average path lengths on a topology whose keys were generated by power-law distribution where $n = 100, 1000, 10000$.

基準が偏ったキー分布や現実的なキー分布においても十分に効果を発揮することを確認した．

今後の課題として，経路長短縮に対する解析的な評価や Skip Graph を拡張した他の構造化オーバレイに対する適用を考えている．

謝辞 本研究は JSPS 科研費 16K12406 の助成を受けたものです．本研究の一部は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務として行われました．本研究は（公財）セコム科学技術振興財団 一般研究助成の支援を受けたものです．

文　献

[1] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications,” IEEE/ACM Trans. Netw., vol.11, no.1, pp.17–32, Feb. 2003.
[2] A. Rowstron and P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems,” Middleware 2001, ed. by R. Guerraoui, pp.329–350, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001.
[3] J. Aspnes and G. Shah, “Skip graphs,” ACM Transactions on Algorithms, vol.3, no.4, p.37, Nov. 2007.
[4] W. Pugh, “Skip lists: A probabilistic alternative to balanced trees,” Communications of the ACM, vol.33, no.6, pp.668–, June 1990.
[5] K. Higuchi, M. Yoshida, N. Miyamoto, and T. Tsuji, “A Routing Algorithm for Distributed Key-Value Store Based on Order Preserving Linear Hashing and Skip Graph,” Applied Computing & Information Technology, ed. by R. Lee, pp.127–139, Studies in Computational Intelligence, Springer International Publishing, 2016.
[6] M. Naor and U. Wieder, “Know Thy Neighbor’s Neighbor: Better Routing for Skip-Graphs and Small Worlds,” Peer-to-Peer Systems III, pp.269–277, Springer, Berlin, Heidelberg, Feb. 2004.
[7] A.G. Beltran, P. Sage, and P. Milligan, “Skip Tree Graph: A Distributed and Balanced Search Tree for Peer-to-Peer Networks,” 2007 IEEE International Conference on Communications, pp.1881–1886, June 2007.
[8] F. Makikawa, T. Tsuchiya, and T. Kikuno, “Balance and Proximity-Aware Skip Graph Construction,” 2010 First International Conference on Networking and Computing, pp.268–271, Nov. 2010.
[9] T. Kawaguchi, R. Banno, M. Hojo, M. Ohnishi, and K. Shudo, “Self-Refining Skip Graph: Skip Graph Approaching to an Ideal Topology,” 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC), pp.441–448, Jan. 2017.
[10] P. Keef and D. Guichard, “Introduction to Higher Mathematics,” `https://www.whitman.edu/mathematics/higher_math_online/` (accessed Jan. 24, 2019). Whitman College.