2-Hop ラベルの直接的な計算によるグラフ最短経路クエリ処理の効率化

秋葉　拓哉† 　岩田　陽一† 　吉田　悠一††

† 東京大学情報理工学系研究科　〒 113–8656 東京都文京区本郷 7-3-1
†† 国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{t.akiba,y.iwata}@is.s.u-tokyo.ac.jp, ††yyoshida@nii.ac.jp

あらまし　2 点間の最短経路の計算は大規模グラフデータにおける最も重要な処理の 1 つであり，グラフデータベースのクエリ処理やネットワークを考慮した情報検索などの幅広い応用を持つ．本論文では (1) グラフに対し予め索引を作成し (2) それを用いて 2 点間の最短経路の問合せに効率的に応答するという問題を扱う．多くの手法が既に提案されているが，今日の規模のグラフデータにおいて満足な性能を持つものは未だに存在していなかった．本論文の提案手法は，索引のデータ構造と問合せ時のアルゴリズムとして 2-Hop と呼ばれる一部の既存手法と共通した枠組みを利用する．しかし，それらの既存手法は索引の計算を最適化問題に帰着し間接的に計算していたのに対し，提案手法は巧妙な枝刈りを伴う幅優先探索により索引を直接的に計算する．実験により既存手法に対し約 100 倍大規模なグラフを同等の前計算時間・問合せ時間で処理できることを確認した．

キーワード　グラフ，ソーシャルネットワーク，ウェブグラフ，最短経路クエリ

1. はじめに

グラフ上の 2 頂点間の最短経路および距離の計算は，グラフにおける最も基礎的な処理の一つであり，様々な幅広い応用を持つ．例えば，ソーシャルネットワークにおいて距離はユーザ間の親密さを表すと考えられ，ソーシャルネットワークを考慮した情報検索 [25,27] や影響力の高い人物及びコミュニティの発見 [14,4] に利用される．ウェブグラフにおいても，距離はページ間の関連性を表すと考えられ，現在観覧中のページとウェブグラフを考慮した検索に利用される [24,16]．他にも，リンクを含むデータベースの top-k キーワード処理 [11,22]，代謝ネットワークにおける最適な代謝経路の発見 [18,19]，コンピュータのネットワークにおけるリソース管理 [15,5] などをはじめとして最短経路や距離は多くの場所で利用される．

最短経路は幅優先探索や Dijkstra のアルゴリズムにより計算できるものの，大規模なネットワークにおいては 1 つの最短経路の計算に数秒や数十秒といった時間を要してしまう．上で述べたような応用の多くでは，最短経路の計算は処理の一部として何度も行われるため，これらは遅すぎる．特に情報検索のような応用では，使用者にできるだけ短い待ち時間で結果を提示したい一方で，1 つの検索結果の順位付けのため多くのペアに関する距離を計算する必要があるため，最短経路の計算は非常に高速であることが求められる．

このため，最短経路を高速に回答するための**索引**を予めグラフに対し構築する**最短経路クエリ問題**に向けた効率的な手法の開発は高い注目を集めており，データ工学のコミュニティを中心として多くの手法が提案されてきている [8,16,26,10,23,3,17,12,25]．しかし，既存の全ての手法は以下のいずれかの問題点を抱えている．

(1) 索引構築のスケーラビリティが低く百万辺規模のグラフしか処理できない [8,26,3,2,12].

(2) 最短経路の応答が他の手法より千倍近く低速である [10,23,17].

(3) 近似手法であり精度に致命的な問題がある [16,25].

そこで本論文では最短経路クエリ問題に対する新たな手法を提案する．提案手法は上の 3 つの問題点の全てを解決する．提案手法は常に正しい最短経路を回答する厳密手法であるため精度の問題は発生しない．また，提案手法は大幅なスケーラビリティの改善を達成し，数億辺規模のグラフに対しても現実的な時間で索引を構築することができる．そして，そのような大規模なグラフにおいても平均質問応答時間は数マイクロ秒であり他の手法に引けを取らない．

提案手法は *2-Hop* [9] と呼ばれる枠組みに基づいているものの，既存手法と全く異なるアプローチで索引を計算する．2-Hop に基づく既存の手法は索引の計算を最適化問題に帰着しそれを解くことにより間接的に索引を計算する [9,8,2,12]．一方，提案手法は全ての頂点から枝刈りを伴う幅優先探索を行い訪れた頂点に距離を記録するという方法により直接的に索引を計算する．ここで用いる枝刈りは，構築途中の索引を用いて質問応答を行うことにより情報が必要か不要かを判断するという巧妙なものになっている．

興味深いことに，提案手法は 2-Hop に基づく手法 [9,8,2,12]，ランドマークに基づく手法 [16,10,23,17]，木分解に基づく手法 [26,3] という有力な既存手法の 3 系統の利点を同時に併せ持つ．まず，提案手法も 2-Hop に基づいているため，質問応答時の計算は索引の 2 つの領域を参照するだけで済み，応答が非常に高速である．また，ランドマークに基づく手法と同様に，提案手法も現実世界のネットワークにハブと呼ばれるような中心性の高い頂点が存在することを効率化のために積極的に活用する．そして，木分解に基づく手法と同様に，提案手法は現実世界のネットワークがコア・フリンジ構造 [6] と呼ばれる構造を持ち次数の低い頂点の周辺が木に近い構造を成していること

表 1: 過去の文献と本論文での実験結果の概要.

手法	グラフ	$\|V\|$	$\|E\|$	索引作成	応答
TEDI	通信	22 K	46 K	17 s	4.2 μs
[26]	ソーシャル	0.6 M	0.6 M	2,226 s	55.0 μs
HCL	ソーシャル	7.1 K	0.1 M	1,003 s	28.2 μs
[12]	引用	0.7 M	0.3 M	253,104 s	0.2 μs
TD	ソーシャル	0.3 M	0.4 M	9 s	0.5 μs
[3]	ソーシャル	2.4 M	4.7 M	2,473 s	0.8 μs
HHL	通信	0.2 M	1.2 M	7,399 s	3.1 μs
[2]	ソーシャル	0.3 M	1.9 M	19,488 s	6.9 μs
PLL	ウェブ	0.3 M	1.5 M	**4 s**	0.5 μs
	ソーシャル	2.4 M	4.7 M	**61 s**	0.6 μs
(提案手法)	ソーシャル	1.1 M	**114 M**	15,164 s	15.6 μs
	ウェブ	7.4 M	**194 M**	6,068 s	4.1 μs

も効率化に活用できる．これらの性質は 4.5 節で証明する．

論文の構成は以下の通りである．2 章で関連研究を紹介する．3 章で背景知識を述べる．4 章で提案手法である枝刈りラベリングアルゴリズムを説明する．5 章で提案手法をビットレベルの並列性を活用し更に高速化する技法を紹介する．6 章で提案手法の拡張について述べる．7 章で実験結果を示す．8 章で本論文をまとめる．

2. 関 連 研 究

2.1 厳 密 手 法

常に正しい最短経路を回答する手法は**厳密手法**と呼ばれる．提案手法も厳密手法である．本節では，既存の厳密手法を紹介する．残念ながら，以下に挙げる全ての厳密手法は索引構築のスケーラビリティに問題があり，たった百万辺規模のグラフの索引を構築するのに少なくとも数千秒から数万秒を要してしまう (表 1).

提案手法を含む多くの効率的な厳密手法は *2-Hop* [9] と呼ばれる枠組みに基づいている．これについては 3.3 節で詳しく説明する．良質な 2-Hop の索引を効率的に計算することは，長年重要な問題として取り組まれてきた [9,8,2,12]．最新の手法の一つである *Hierarchical Hub Labeling* [2] は道路ネットワークに向けて開発された手法である *Hub Labeling* [1] に基づいている．別の 2-Hop に関連する最新の手法として，全域木を活用する *Highway-Centric Labeling* [12] も提案されている． その他，**木分解** [20] を用いることでネットワークのコア・フリンジ構造 [6] を陽に活用する手法なども提案されている [26,3].

2.2 近 似 手 法

より大規模なネットワークを処理するため，最短経路とは限らない短い経路を近似として回答する手法である**近似手法**もよく研究されてきた．しかし，以下で説明する通りこれらにも精度か質問応答時間のいずれかの問題点がある．

近似手法の大部分は**ランドマーク**に基づく手法である．これらの手法は，小さな頂点集合 L をランドマークとして選択し，それらの頂点から全点への最短経路を前計算し索引とする [21,25] ．距離を回答する際には L のいずれかの頂点を経由する経路により最短経路を近似する．この手法の精度は正しい最短経路が L や L の近くを通過するほど有利となる．従って，ハブと呼ばれるような中心的な頂点をランドマークとして選択することで，ランダムな質問に対してはかなり高い精度を達成する [16,7]．しかし，近い頂点対に関する精度はそれらの平均精度に比べ極めて悪いということが後で指摘されている [17,3]. 情報検索などの応用では，距離は近い頂点対を識別するために使われるため，この問題点は致命的である．

そこで，索引から得た経路をもとに閉路や近道の検出を行い更に短い経路を構成し回答することで精度を向上させる技法が開発された [10,23,17]．それらの技法は確かに精度を向上させるものの，質問応答時間は厳密手法を含む他の手法と比べ約 1000 倍程度低速になってしまう．

3. 前 準 備

3.1 表 記

本論文ではグラフ $G = (V, E)$ として表されるネットワークを扱う．グラフ G の頂点数を n，辺数を m と表す．頂点 $v \in V$ の近傍を $N_G(v)$ と表す．即ち，$N_G(v) = \{u \in V \mid (u, v) \in E\}$ である．$d_G(u, v)$ は 2 頂点 u, v の距離を示す．u から v が到達不能である場合 $d_G(u, v) = \infty$ と定義する． また，$P_G(u, v) \subseteqq V$ を 2 頂点 u, v 間の最短経路に含まれる頂点の集合とする．最短経路が複数存在する場合，全ての最短経路に関する和集合とする．即ち，$P_G(u, v) = \{w \in V \mid d_G(u, w) + d_G(w, v) = d_G(u, v)\}$ である．

3.2 問 題 設 定

本論文で扱う最短経路クエリ問題は，以下のように索引構築と質問応答の 2 つのステップからなる．1 つのグラフに対して，索引構築は 1 回だけ行い，構築した索引を用いて質問応答を繰り返し行う．

(1) 索引構築 — グラフ G を受け取り索引を構築する．
(2) 質問応答 — 頂点の組 u, v を受け取り距離 $d_G(u, v)$ を索引を用い回答する．

説明を簡単にするため，最短経路ではなく距離を回答するものとした．また，以下ではまずグラフ G を重みなしの無向グラフであると仮定して説明を行う．ただし，実際には最短経路の回答，重みの対応，有向グラフの対応は全て可能であり，それらについては 6 章で議論する．

3.3 2-Hop の枠組み

2-Hop [9] の枠組み，即ち索引のデータ構造と応答のアルゴリズムを以下で説明する．我々の提案手法もこの枠組みに基づいている．

索引として計算し保存するのは，各頂点 v に関する**ラベル** $L(v)$ である．各ラベル $L(v)$ は，u は頂点，$\delta_{uv} = d_G(u, v)$ であるような組 (u, δ_{uv}) の集合である．全頂点に対するラベル $\{L(v)\}_{v \in V}$ が**索引**である．

2 頂点 s, t 間の距離が質問された時，以下のように定義される $\textsc{Query}(s, t, L)$ を計算し応答する．

$$\textsc{Query}(s, t, L) = \min\{\delta_{vs} + \delta_{vt} \mid (v, \delta_{vs}) \in L(s), (v, \delta_{vt}) \in L(t)\}.$$

$L(s)$ と $L(t)$ に共通する頂点が存在しない時は $\textsc{Query}(s, t, L) = \infty$ と定義する．任意の 2 頂点 s, t について $\textsc{Query}(s, t, L) = d_G(s, t)$ となる時，L は G の**正しい** *2-Hop* **索引**と呼ばれる．

各頂点について $L(v)$ を頂点をキーとして整列しておくこ

とで，$\textsc{Query}(s,t,L)$ はマージソートのマージの操作と同様に $O(|L(s)|+|L(t)|)$ 時間で計算することができる．

この枠組みに基づく正しくかつ小さい 2-Hop 索引を効率的に計算するアルゴリズムを開発することが長年取り組まれてきた課題である．例えば，各ラベル $L(v)$ に全点への距離を覚えるような索引は，自明な正しい 2-Hop 索引であるが，サイズが $\Theta(n^2)$ であり使い物にならない．また，7 章の実験結果でも確認するように，現実のネットワークにはそれより遥かに小さい 2-Hop 索引が存在する．従って，索引が正しいことを保証しながら，索引に距離を保存しない頂点対を選択していかなければならない．そして，大規模なネットワークを処理するためには，それを非常に効率的に行わなければならない．

4. 提 案 手 法

この章では，正しく小さい 2-Hop 索引を効率的に計算するための提案手法である**枝刈りラベリングアルゴリズム**を説明する．説明は，まず枝刈りを行わない素朴なアルゴリズムを紹介しそこに枝刈りを導入するという 2 つのステップで行う．その後で，手法の正当性や現実のネットワークの性質を活用する方法について議論する．

4.1 素朴なラベリングアルゴリズム

まず，枝刈りを行わない素朴な手法から説明する．この手法は，全頂点から幅優先探索 (BFS) を行い，そこで得られた距離の全てをラベルとして保存するものである．自明かつ非効率的な手法であるものの，提案手法をこれに基づき説明するため，以下でより詳しく説明する．

頂点集合を $V=\{v_1,v_2,\ldots,v_n\}$ とする．全頂点 u について $L_0(u)=\emptyset$ であるような空の索引 L_0 から開始する．頂点 $v_1,v_2,\ldots,v_n$ の順で始点を選び BFS を行うものとする．即ち，k 回目の BFS は頂点 v_k を始点とする．v_k からの BFS の後，到達した全ての頂点のラベルに距離を追加する．即ち，$d_G(v_k,u)\neq\infty$ である全頂点 u について $L_k(u)=L_{k-1}(u)\cup\{(v_k,d_G(v_k,u))\}$ として新たなラベル $L_k(u)$ を得る．到達できなかった頂点，即ち $d_G(v_k,u)=\infty$ であるような頂点 u に関しては $L_k(u)=L_{k-1}(u)$ とする．v_k からの BFS で得られた距離は $L(v_k)$ に追加するのではなく到達したそれぞれの頂点のラベルに追加されることに注意されたい．

このようにして構築した L_n が最終的な索引である．L_n は正しい 2-Hop 索引であるが，$\Theta(n^2)$ のサイズを持つこの 2-Hop 索引は，明らかにサイズが大きすぎ，現実的ではない．

補題 4.1. 任意の二頂点 $s,t\in V$ について $\textsc{Query}(s,t,L_n)=d_G(s,t)$.

4.2 枝刈りラベリングアルゴリズム

次に，枝刈りを上の素朴な手法に導入することで提案手法である枝刈りラベリングアルゴリズムを説明する．先程と同様に，頂点 $v_1,v_2,\ldots,v_n$ の順で始点を選び枝刈り BFS を行う．空の索引 L'_0 から開始し，頂点 v_k からの枝刈り BFS を行った後 L'_{k-1} の一部に組を追加し L'_k を得るということを繰り返し，L'_n を最終的な索引とする．即ち，先の素朴なアルゴリズムと異なる点は，通常の BFS の代わりに枝刈り BFS を行うという点のみである．しかし，7.2.1 節などで確認するように，その差異は驚くほど性能を向上させる．

Algorithm 1 頂点 v_k からの枝刈り BFS.

```
1: procedure PrunedBFS(G, v_k, L'_{k-1})
2:     Q ← v_k のみを含むキュー
3:     P[v_k] ← 0 かつ P[v] ← ∞ for all v ∈ V(G) \ {v_k}.
4:     L'_k[v] ← L'_{k-1}[v] for all v ∈ V(G).
5:     while Q が空になるまで do
6:         u を Q から取り出す
7:         if Query(v_k, u, L'_{k-1}) ≦ P[u] then
8:             continue
9:         L'_k[u] ← L'_{k-1}[u] ∪ {(v_k, P[v_k])}
10:        for all w ∈ N_G(v) s.t. P[w] = ∞ do
11:            P[w] ← P[u] + 1.
12:            w を Q に追加
13:    return L'_k
```

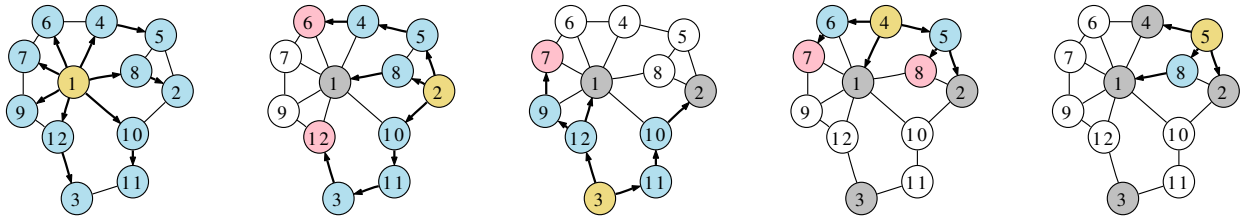
Algorithm 2 提案手法による 2-Hop 索引 L の計算.

```
1: procedure Preprocess(G)
2:     L'_0[v] ← ∅ for all v ∈ V(G).
3:     for k = 1, 2, ..., n do
4:         L'_k ← PrunedBFS(G, v_k, L'_{k-1})
5:     return L'_n
```

BFS の枝刈りは以下のように行う．索引 L'_{k-1} を持っており頂点 v_k からの BFS を行いながら L'_k を作成しているとしよう．現在，頂点 u に距離 δ で訪れていると仮定する．枝刈りの判定を行うにあたって，作りかけの索引 L'_{k-1} に質問を発行し，$\textsc{Query}(v_k,u,L'_{k-1})$ を計算する．そして，もし $\textsc{Query}(v_k,u,L'_{k-1})\leqq\delta$ であれば頂点 u を枝刈りする．即ち，組 (v_k,δ) をラベル $L'_k(u)$ に追加せず (つまり $L'_k(u)=L'_{k-1}(u)$ とする)，また頂点 u からの枝を無視する．そうでなければ，通常通り，$L'_k(u)=L'_{k-1}(u)\cup\{(v_k,\delta)\}$ のように組を追加し，u からの全ての辺を走査する．先の素朴な手法と同様に，幅優先探索で到達しなかった頂点 u についても $L'_k(u)=L'_{k-1}(u)$ とする．枝刈り幅優先探索のアルゴリズムは Algorithm 1，全体としての前処理アルゴリズムは Algorithm 2 として示されている．

この枝刈りの正当性は非自明である．例えば，現在の索引 L'_{k-1} で v_k,u 間の距離が正しく答えられるからといって，$L'_k(u)$ に組 (v_k,δ) が追加されないと，u から別の頂点 w への距離 $d_G(u,w)$ が正しく答えられなくなってしまうかもしれない．あるいは，u に関しては組 (v_k,δ) を追加しなくてもよかったとしても，u からの辺を全て無視する影響で BFS が到達しない頂点に関して問題が発生するかもしれない．こういった懸念がなく構築される索引は常に正しい 2-Hop 索引になるということは，次の節 4.3 で証明する．

例: 図 1 は枝刈りラベリングアルゴリズムの例を示している．最初の枝刈り BFS は頂点 1 から行われ，全ての頂点を訪れる (図 1a)．次の枝刈り BFS は頂点 2 から行われる (図 1b)．頂点 6 に到達した時，$\textsc{Query}(2,6,L'_1)=d_G(2,1)+d_G(1,6)=3=d_G(2,6)$ であるので頂点 6 は枝刈りされ，そこからの辺は無視される．頂点 1, 12 も同様に枝刈りされる．アルゴリズムが進むにつれ，枝刈りは頻繁に生じ，探索空間はどんどん小さくなる (図 1c,1d,1e).



(a) まず頂点 1 からの BFS を行う．全ての頂点を訪れる．(b) 次に頂点 2 からの BFS を行う．頂点 1, 6, 7, 9, 12 にはラベルが追加されない．(c) 次に頂点 3 からの BFS を行う．下半分の頂点にしか訪れない．(d) 次に頂点 4 からの BFS を行う．今度は上半分の頂点にしか訪れない．(e) 次に頂点 5 からの BFS を行う．探索空間は更に小さくなる．

図 1: 枝刈りラベリングアルゴリズムの動作例．黄色の頂点は根を，青い頂点は到達しラベルが追加された頂点を，赤の頂点は到達したが枝刈りされた頂点を，灰色の頂点は既に根となった頂点を表す．

計算量: 正確な時間計算量の解析は困難であるため，見積もりを示す．l を平均ラベルサイズとする．合計で $O(nl)$ 頂点に到達し，各頂点で平均 $O(m/n)$ 辺を走査し $O(l)$ 時間で枝刈り判定を行う．従って，全体の時間計算量は $O(ml + nl^2)$ 時間であると大まかに見積もれる．この見積もりは 7 章で示される実験結果と大まかに一致している．また，実験結果により l は数百程度であることがわかる．

4.3 正 当 性

上の枝刈りラベリングアルゴリズムが正しい 2-Hop 索引を計算することを証明する．即ち，任意の二頂点 s, t について $\textsc{Query}(s, t, L'_n) = d_G(s, t)$ となることを示す．以下，L と L' はそれぞれ素朴なアルゴリズムと枝刈りラベリングによって構築された索引であることに注意せよ．

補題 4.2. $0 < k \leqq n$ とし，任意の 2 頂点 s, t について $\textsc{Query}(s, t, L'_{k-1}) = \textsc{Query}(s, t, L_{k-1})$ と仮定する．この時，ある頂点 $u \in V$ について $(v_k, \delta_{v_k u}) \in L'_k(u)$ となる必要十分条件は，$i < k$ なる任意の i について $v_i \notin P_G(v_k, u)$ であることである．

証明. まず，頂点 $u \in V$ について，ある $i < k$ が存在し $v_i \in P_G(v_k, u)$ である時，$\textsc{Query}(v_k, u, L'_{k-1}) = d_G(v_k, u)$ となるため，頂点 v_k からの BFS では頂点 u は枝刈りされ，$(v_k, \delta_{v_k u}) \notin L'_k(u)$ となる．

次に，頂点 $u \in V$ について，任意の $i < k$ で $v_i \notin P_G(v_k, u)$ であると仮定する．頂点 v_k からの BFS を行なっている最中であると仮定し，頂点 $w \in P_G(v_k, u)$ に訪問しているとする．$P_G(v_k, w) \subseteqq P_G(v_k, u)$ であるため，任意の $i < k$ について $v_i \notin P_G(v_k, w)$ である．従って $\textsc{Query}(v_k, w, L'_{k-1}) > d_G(v_k, w)$ となり，頂点 w は枝刈りされない．よって，$P_G(v_k, u)$ 内の全ての頂点で枝刈りは起こらず，頂点 v_k からの BFS は頂点 u に到達し，$(v_k, \delta_{v_k u}) \in L'_k(u)$ となる． □

定理 4.1. 任意の $0 \leqq k \leqq n$ と二頂点 $s, t \in V$ について $\textsc{Query}(s, t, L'_k) = \textsc{Query}(s, t, L_k)$.

証明. k に関する数学的帰納法により証明を行う．$L'_0 = L_0$ であるため $k = 0$ の時定理は成立する．以下では $0, 1, \ldots, k-1$ における主張の成立を仮定し k における成立を示す．

$s, t \in V$ をグラフのある 2 頂点とする．それらが到達不可能である時，距離 ∞ は明らかに得られるため，以下ではそれらは到達可能であると仮定する．j を $(v_j, \delta_{v_j s}) \in L_k(s), (v_j, \delta_{v_j t}) \in L_k(t)$ かつ $\delta_{v_j s} + \delta_{v_j t} = \textsc{Query}(s, t, L_k)$ となるような最小の j とする．すると，任意の $i < j$ について $v_i \notin P_G(v_j, s)$ かつ $v_i \notin P_G(v_j, t)$ であるため，補題 4.2 より，$(v_j, \delta_{v_j s})$ と $(v_j, \delta_{v_j t})$ は $L'_k(s)$ と $L'_k(t)$ にも含まれる．従って，$\textsc{Query}(s, t, L'_k) = \textsc{Query}(s, t, L_k)$ である． □

$k = n$ としてこの定理を用いることにより，提案手法の正当性が証明できる．

系 4.1. 任意の二頂点 $s, t \in V$ について $\textsc{Query}(s, t, L'_n) = d_G(s, t)$.

4.4 枝刈り BFS を行う頂点の順番

ここまでの説明では，枝刈り BFS の根は $v_1, v_2, \ldots, v_n$ の順番で選択してきたが，この順番は自由であり選択の余地がある．そして実際には，この順番は提案手法の性能に大きく影響する．これは，枝刈りの効率が，そこまでで構築された索引でどれだけの頂点対に関して正しい最短経路を回答できるかに依存し，また現実のネットワークでは頂点の中心性に大きな開きがあるためである．中心的な頂点から順に選ぶことで，枝刈りの効率は大きく向上する．

これは，2.2 節で紹介したランドマークに基づく手法におけるランドマークの頂点を選ぶ方法の議論 [16] とかなり類似している．そこでの知見をもとに，本論文では，頂点を次数の大きい順に選択することにする．7 章の実験結果で確認する通り，この方法は簡潔ながらかなり高い性能を達成する．しかし，より洗練された頂点の順番付けの方法を模索することは今後の課題となり得る．

4.5 理論的な性質

枝刈りラベリングアルゴリズムは上で述べた通り極めて簡潔であり，アルゴリズム自体は別段現実のネットワークの性質を陽に活用しようとするものではない．しかし，興味深いことに，枝刈り BFS を行う順番の工夫によってハブの存在やフリンジ部分の小さい木幅などの現実のネットワークの性質を有利に活用することができる．以下では提案手法のそのような性質について議論する．

4.5.1 ハブの活用

4.4 節で述べた通り，提案手法は，中心性の高いハブのような頂点から優先的に枝刈り BFS を行うことにより枝刈りの効率を著しく高めることができ，ハブを活用しているという点でランドマークに基づく既存手法と共通の性質を持つ．ランド

マークに基づく手法として，パスに関するヒューリスティクスを用いない素朴な手法 [16,25] を考える．この時，ランドマークに基づく手法の精度と提案手法のラベルサイズにおいて以下のような定理を示すことができる．

定理 4.2. ランドマークに基づく近似手法が $(1-\epsilon)n^2$ 個以上の頂点対 (n^2 個中) に関して k 個のランドマークで正しい距離を回答できるとき，提案手法は平均ラベルサイズ $O(k+\epsilon n)$ の索引を計算できる．

証明. k 個のランドマークから枝刈り BFS を行った後には，現在の索引で正しく距離を回答できない高々 ϵn^2 個の組が索引に追加される． □

4.5.2 フリンジ部分の小さい木幅の活用

次に，木分解に基づく既存手法 [26,3] と同様に提案手法がフリンジ部分の木に近い構造を活用することができるという点に関して議論する．現実世界のネットワークはクラスタ性などとして知られる性質により次数の低い部分は木に近い構造をなしている [6]．木に近いことから小さい木幅を期待しこれを陽に活用するのが木分解を用いる手法である．一方，提案手法は木分解を陽に用いないものの，頂点の順番の工夫により小さい木幅を活用できることが以下のように証明できる．このことは，提案手法もフリンジ部分の木に近い構造を活用していることを示唆する．木分解や木幅の定義については [20] を参照せよ．

定理 4.3. グラフ G の木幅を w としたとき，提案手法は $O(wm\log n + w^2 n\log^2 n)$ 時間で前処理を行い，索引のサイズは $O(wn\log n)$ となり，質問に $O(w\log n)$ 時間で応答できる．

証明. 木分解の重心分解 [13] を考える．重心であるバッグの全頂点から枝刈り BFS が行われた以降は，枝刈り BFS はそのバッグを超えて行われない．従って，木分解を分割して考えてよく，各木分解のバッグの個数は半分以下になっている．これを再帰的に考えると，再帰の深さは $O(\log n)$ となる．再帰の各ステップにおいて各頂点のラベルには高々 w 個の組が追加されるため，各ラベルのサイズは $O(w\log n)$ である．また，再帰の各ステップにおいて辺の走査が $O(wm)$ 時間，枝刈り判定が $O(w^2 n\log n)$ 時間となるため，全体の計算量は $O(wm\log n + w^2 n\log^2 n)$ となる． □

5. ビットレベル並列性を利用した高速化

索引作成や質問応答をさらに効率化するため，**ビットレベル並列性**を活用する提案手法の高速化法を提案する．ビットレベル並列性とは，コンピュータの 1 ワード内の異なるビットで異なる計算を同時に進めることができるという並列性である．現代のコンピュータの 1 ワードは 32 ビットや 64 ビットであるため，32 個あるいは 64 個の異なる計算を同時に行うことができる．(この並列性はスレッドレベル並列性とは異なることに注意されたい．)

以下において，コンピュータの 1 ワード内のビット数を b とし，b ビットのビットベクトルの演算を $O(1)$ 時間で行えると仮定する．以下では，$b+1$ 頂点からの BFS によるラベルの計算をビット並列性を活用して一気に $O(m)$ 時間で行う手法，及びそのラベルの保存法を工夫して質問時にそれら $b+1$ 頂点のいずれかを経由する距離を $O(1)$ 時間で計算する方法を提案する．

5.1 索引のデータ構造

索引作成や質問応答のアルゴリズムを説明するため，まず，索引に保存するデータを定義する．効率化のため，索引のデータは今までの通常の 2-Hop ラベルとは異なる形式で保存される．次の節で説明する通り，ビットベクトル BFS はある頂点 r とサイズ b 以下の r の近傍の部分集合 $S_r \subseteqq N_G(r)$ から一気に行われる．このとき，以下の補題はビット並列性の活用のために重要な洞察である．

補題 5.1. 任意の頂点 $u \in S_r$ と頂点 $v \in V$ について，$|d_G(u,v) - d_G(r,v)| \leqq 1$.

$S_r^i(v)$ を $S_r^i(v) = \{u \in S_r \mid d_G(u,v) - d_G(r,v) = i\}$ と定義する．上の補題より，各頂点 $v \in V$ について，S_r は 3 つの集合 $S_r^{-1}(v), S_r^0(v), S_r^{+1}(v)$ に分割される．

索引として，各頂点 $v \in V$ についてラベル $L_{\mathrm{BP}}(v)$ を計算し保存する．$L_{\mathrm{BP}}(v)$ はタプル $(u, \delta_{uv}, S_u^{-1}(v), S_u^0(v))$ の集合であり，$u \in V$ は根の頂点を表し，$\delta_{uv} = d_G(u,v)$ かつ $S_u^i(v) \subseteqq S_u$ は上の定義の通りである．$S_u^{-1}(v)$ と $S_u^0(v)$ は b ビットのビットベクトルで保存する．$S_u^{+1}(v)$ は保存せずとも $S_u \setminus (S_u^{-1}(v) \cup S_u^0(v))$ として得られるため保存しない．これを通常のラベルと区別し**ビットベクトルラベル**と呼ぶことにする．

5.2 索引構築アルゴリズム

次に索引計算のアルゴリズムを説明する．本節ではまず，4.2 節で導入した枝刈りは考えず，4.1 節で紹介した素朴なラベリングアルゴリズムをビットレベル並列性を活用するように改良する．枝刈りとの併用は 5.4 節にて議論する．

$r \in V$ を根の頂点，$S_r \subseteqq N_G(r)$ をサイズ b 以下の r の近傍の部分集合とする．以下では，$\{r\} \cup S_r$ から到達可能な全ての頂点 $v \in V$ について $d_G(r,v)$, $S_r^{-1}(v)$ 及び $S_r^0(v)$ を計算するアルゴリズムを説明する．基本的なアイディアとしては，r から BFS を行いながら，集合 S^{-1} と S^0 を動的計画法とビット演算により計算する．

v をある頂点とし，$d_G(r,w) < d_G(r,v)$ なる全ての頂点 w について $S_r^{-1}(w)$ が既に計算されているとする．この時，$S_r^{-1}(v)$ は以下のように計算できる．

$$\left\{u \in S_r \mid u \in S_r^{-1}(w), w \in N_G(v), d_G(r,w) = d_G(r,v) - 1\right\}.$$

これは，u がもし $S_r^{-1}(v)$ に入っていれば，$d_G(u,v) = d_G(r,v) - 1$ であり，従って u は r から v への最短経路の 1 つに含まれているからである．同様に，$d_G(r,w) \leqq d_G(r,v)$ なる全ての w について $S_r^{-1}(w)$ が，及び $d_G(r,w) < d_G(r,v)$ なる全ての w について $S_r^0(w)$ が既に得られている時，$S_r^0(v)$ は以下のように計算できる．

$$\left\{u \in S_r \mid u \in S_r^0(w), w \in N_G(v), d_G(r,w) = d_G(r,v) - 1\right\}$$
$$\cup \left\{u \in S_r \mid u \in S_r^{-1}(w), w \in N_G(v), d_G(r,w) = d_G(r,v)\right\}.$$

従って，BFS に伴って動的計画法を行い，S_r^{-1} と S_r^0 を r からの距離の昇順で計算できる． 集合に関する処理は全てビット演算により $O(1)$ 時間で行えるため，1 回のビットベクトル BFS は全体で $O(m)$ 時間で行える．

5.3 質問応答アルゴリズム

2 頂点 s,t 間の距離を回答するにあたって，通常のラベルを用いる時と同様に，ビットベクトルラベル $L_{\mathrm{BP}}(s)$ と $L_{\mathrm{BP}}(t)$ を走査する．そして，根の頂点が等しい 2 つのタプル $(r,\delta_{rs},S_r^{-1}(s),S_r^0(s)) \in L_{\mathrm{BP}}(s)$ と $(r,\delta_{rt},S_r^{-1}(t),S_r^0(t)) \in L_{\mathrm{BP}}(t)$ から，$\{r\} \cup S_r$ の少なくとも 1 つ以上の頂点を経由する s,t 間の最短経路の長さを計算する．即ち，$\delta = \min_{u\in\{r\}\cup S_r}\{d_G(s,u)+d_G(u,t)\}$ を計算する．全てのタプルをこのように調べ，それらの最小値を距離として答える．

素朴な方法は，ビットベクトルから $d_G(s,u)$ と $d_G(u,t)$ を全ての $u \in S_r$ について復元し素直に計算する方法である．この方法では，1 つの組を処理するのに $O(|S_r|)$ 時間を必要とする．しかし，以下ではビット演算を活用し δ を $O(1)$ 時間で計算する方法を提案する．

$\tilde{\delta} = d_G(s,r)+d_G(r,t)$ とおく．$\tilde{\delta}$ は δ の上界であり，任意の $u \in S_r$ について $d_G(s,u) \geqq d_G(s,r)-1$ かつ $d_G(u,t) \geqq d_G(r,t)-1$ であるため，$\tilde{\delta}-2 \leqq \delta \leqq \tilde{\delta}$ である．従って，行うべきことは，距離 δ が $\tilde{\delta}-2, \tilde{\delta}-1, \tilde{\delta}$ のいずれであるかを判定することである．

これは，以下のように行える．もし $S_r^{-1}(s) \cap S_r^{-1}(t) \neq \emptyset$, であれば $\delta = \tilde{\delta}-2$ である．そうでなくて，もし $S_r^0(s) \cap S_r^{-1}(t) \neq \emptyset$ であるか $S_r^{-1}(s) \cap S_r^0(t) \neq \emptyset$ であれば $\delta = \tilde{\delta}-1$ である．そして，それ以外の場合は，$\delta = \tilde{\delta}$ である．積集合の計算はビットごとの AND 演算により行うことができるため，これらの判定は $O(1)$ 時間で行うことができる．従って，距離 δ は $O(1)$ 時間で計算でき，全体で，各クエリに $O(|L_{\mathrm{BP}}(s)|+|L_{\mathrm{BP}}(t)|)$ 時間で回答できる．

5.4 枝刈りラベリングとの併用

最後に，以上のビットレベル並列性を活用する手法を，4.2 節で導入した枝刈りラベリングと併用し提案手法の性能を向上させる方法について議論する．上のビットベクトル BFS に同様の枝刈りを導入するという方法もあり得るものの，提案するのは，最初に枝刈り無しのビットベクトル BFS を何度か行い，次に枝刈りラベリングを行うという方法であり，こちらのほうが簡潔かつ実用的には効果的である．即ち，t をパラメータとし，ビットベクトル BFS を t 回最初に行い $(b+1)t$ 頂点に関するビットベクトルラベルを計算し，次に残りの $n-(b+1)t$ 頂点から枝刈り BFS を行い通常のラベルを計算する．根 r は同様に最も次数のものから選んでいき，集合 S_r はその近傍であり未選択のものを次数の高いものから選ぶようにする．

この方法は枝刈りラベリングとビットベクトルラベリングの長所を組み合わせた手法になっている．索引構築の序盤では，索引は小さいため，枝刈りはあまり効果的に働かない．そこで，ビットベクトルラベリングにより多くの頂点に関するラベルを一度に計算することで，その部分を高速化することができる．

7 章で示す通り，ビットベクトルラベリングは性能を大きく向上する．また，パラメータ t に関する実験も行なっている．特に，提案手法の性能は t の値に敏感ではなく，極端に大きすぎる値を与えない限り性能は安定しているため，パラメータ t を伴っても提案手法は依然として簡単に利用することができる．

表 2: データセットの情報 (K=10^3, M=10^6).

データセット	種類	$\lvert V\rvert$	$\lvert E\rvert$
Gnutella	通信	63 K	148 K
Epinions	ソーシャル	76 K	509 K
Slashdot	ソーシャル	82 K	948 K
Notredame	ウェブ	326 K	1.5 M
WikiTalk	ソーシャル	2.4 M	4.7 M
Skitter	通信	1.7 M	11 M
Indo	ウェブ	1.4 M	17 M
MetroSec	通信	2.3 M	22 M
Flickr	ソーシャル	1.8 M	23 M
Hollywood	ソーシャル	1.1 M	114 M
Indochina	ウェブ	7.4 M	194 M

6. 拡　　張

最短経路クエリ: 距離だけでなく最短経路を得たい場合，$p_{uv} \in V$ を u からの BFS 木における v の親として，ラベル $L(v)$ にタプル (u,δ_{uv},p_{uv}) を保存するようにする．最短経路を得るには，2 点から BFS 木の親を辿っていけばよい．

重み付きグラフ: 重み付きグラフを扱うには，枝刈り BFS と同様の枝刈りを Dijkstra のアルゴリズムに導入すればよい．

有向グラフ: 有向グラフを扱うには，順向きの距離に関するラベル $L_{\mathrm{OUT}}(v)$ と逆向きの距離に関するラベル $L_{\mathrm{IN}}(v)$ を計算し保持するようにすれば良い．

7. 評 価 実 験

本実験には CPU が Intel Xeon X5670 (2.93 GHz)，メモリが 48GB の Linux サーバを利用した．全てのアルゴリズムは C++ で実装された．うち，提案手法の実装は公開している(注1)．ビットベクトルは 64 ビットとした．質問応答時間は 1,000,000 回のランダムな質問に対する平均時間を報告している．

実験には，現実世界のネットワークから，5 つのソーシャルネットワーク，3 つのウェブグラフ，3 つの通信ネットワークを用いた．全てのネットワークを無向・重みなしのネットワークとして扱った．各ネットワークの種類，頂点数および辺数は表 2 にまとめられている．

7.1 性能の検証

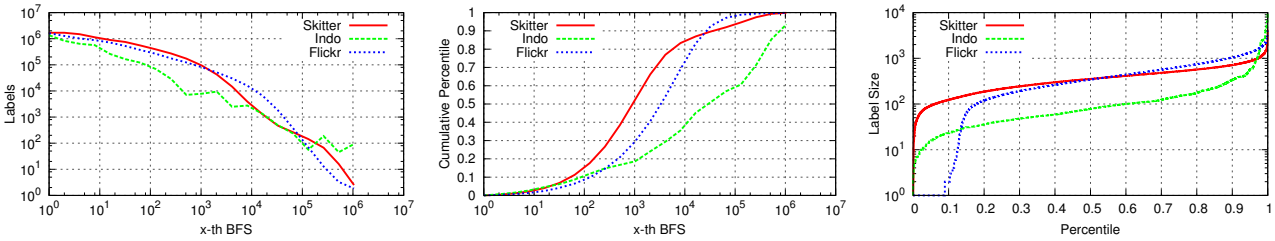
まず，効率や頑健性を検証するため，実データにおける提案手法の性能を既存手法と比較する．表 3 はその結果を示している．平均ラベルサイズは (通常のラベルの数) + (ビット並列ラベルの数) という書式で記している．ビットベクトル BFS の回数 t は，最初の 5 つのデータセットに関しては 16，後半 6 個のデータセットに関しては 64 と設定した．

2 つの既存手法を比較対象として選択した．1 つ目は Hierarchical Hub Labeling [2] であり，2-Hop に基づく既存手法の中で最も性能の良いものである．2 つ目は木分解に基づく手法 [3] である．これは，TEDI [26] を改良した手法である．これらの手法が既存手法の中で特に高性能なものであるということは表 1 より確認できる．

(注1)：http://git.io/pll

表 3: 既存手法と提案手法の実データを用いた性能比較．IT は索引構築時間を，IS は索引サイズを QT は平均質問応答時間を，LN は平均ラベルサイズを表す．DNF は 1 日間で計算が終わらなかったかメモリ不足により実行不可能であったことを表す．

データセット	提案手法				Hierarchical Hub Labeling [2]				Tree Decomposition [3]			BFS
	IT	IS	QT	LN	IT	IS	QT	LN	IT	IS	QT	
Gnutella	54 s	209 MB	5.2 μs	644+16	245 s	380 MB	11 μs	1,275	209 s	68 MB	19 μs	3.2 ms
Epinions	1.7 s	32 MB	0.5 μs	33+16	495 s	93 MB	2.2 μs	256	128 s	42 MB	11 μs	7.4 ms
Slashdot	6.0 s	48 MB	0.8 μs	68+16	670 s	182 MB	3.9 μs	464	343 s	83 MB	12 μs	12 ms
NotreDame	4.5 s	138 MB	0.5 μs	34+16	10,256 s	64 MB	0.4 μs	41	243 s	120 MB	39 μs	17 ms
WikiTalk	61 s	1.0 GB	0.6 μs	34+16	DNF	-	-	-	2,459 s	416 MB	1.8 μs	197 ms
Skitter	359 s	2.7 GB	2.3 μs	123+64	DNF	-	-	-	DNF	-	-	190 ms
Indo	173 s	2.3 GB	1.6 μs	133+64	DNF	-	-	-	DNF	-	-	150 ms
MetroSec	108 s	2.5 GB	0.7 μs	19+64	DNF	-	-	-	DNF	-	-	150 ms
Flickr	866 s	4.0 GB	2.6 μs	247+64	DNF	-	-	-	DNF	-	-	361 ms
Hollywood	15,164 s	12 GB	15.6 μs	2,098+64	DNF	-	-	-	DNF	-	-	1.2 s
Indochina	6,068 s	22 GB	4.1 μs	415+64	DNF	-	-	-	DNF	-	-	1.5 s

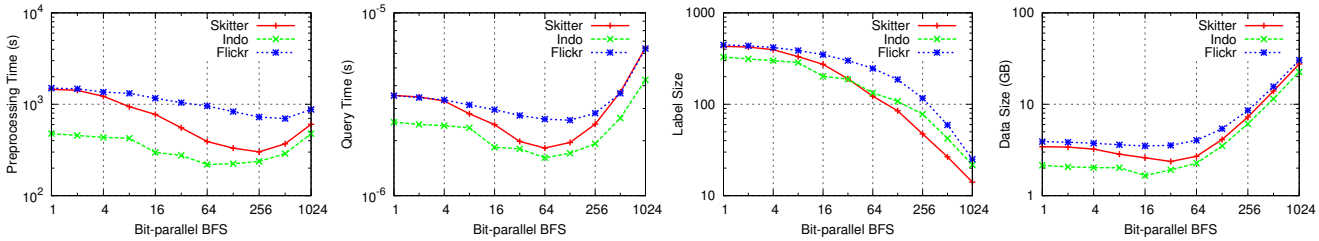


(a) 各 BFS においてラベルに組が追加された頂点の数.

(b) 各 BFS においてラベルに組が追加された頂点の数の累積分布.

(c) 各頂点のラベルサイズの分布.

図 2: 枝刈りの効果およびラベルのサイズ.



(a) 索引構築時間

(b) 平均質問応答時間

(c) 通常のラベルの平均サイズ

(d) 索引サイズ

図 3: ビットベクトル BFS の回数 t に対する性能の変化.

7.1.1 索引構築時間とスケーラビリティ

我々が強調したい結果は，索引構築時間の大幅な短縮によるスケーラビリティの向上である．まず，数億辺・数百万頂点からなる 2 つのデータセット Hollywood と Indochina の索引を数時間で構築することができた．表 1 からもわかる通り，既存手法は数百万辺のグラフを処理の限界としていたため，提案手法は一気に約 100 倍のスケーラビリティ改善を達成したことになる．

数千万辺を持つ次の 4 つのデータセットに関しても，提案手法は千秒以下で索引の作成が終了したのに対し，既存手法は 1 日以上かかってしまうかメモリ不足かの要因で索引の作成を行うことができなかった．更に小さなデータセットに関しては既存手法も索引の作成を行うことができたが，やはり提案手法のほうが約 50 から 100 倍程度高速であることがわかる．

7.1.2 応 答 時 間

質問への平均応答時間は，一般に数マイクロ秒であり，最も遅いデータセットであっても約 16 マイクロ秒であった．小さい方から 5 つのデータセットに対する結果から，提案手法は応答時間に関しても既存手法より高速であることが伺える．また，より大きなグラフに対する結果より，グラフの規模が大きくなっていっても応答時間の増加は緩やかであり大規模なグラフでも高速な応答を実現できるという事もわかる．

7.1.3 索引サイズ

小さい方から 5 つのデータセットに対する結果より，提案手法の索引サイズは既存手法と比べ少し良いか互角程度であるということがわかる．大規模なグラフに関する索引のサイズはやや大きくなる．数十 GB のメモリを搭載するコンピュータは今となっては珍しくもなく特別高価でもないため，これは大きな問題とならないものの，索引サイズを減らすことはより大規模なグラフに向けた次の課題としたい．

7.2 分 析

次に，提案手法の動作やパラメータに関する性能の変化等について調べる．

7.2.1 枝 刈 り

図 2a は各枝刈り BFS にて追加された距離の情報の数を表している．この図から，提案手法の枝刈りが非常に効果的であ

ることが読み取れる．例えば，99% にあたる一万回目以降の BFS ではグラフの 1% 以下にしか訪れないなどといったことが分かる．この計測ではビットベクトルラベリングを併用していないため，実際には広域的な BFS が行われる回数は更に少ない．また，図 2b はその累積分布である．索引の殆どは序盤に作られるという事がわかり，ハブの頂点からの最初のラベルが非常に効果的であるということがわかる．

7.2.2 ラベルサイズ

図 2c は索引における各頂点のラベルサイズの分布を表している．この図より，ラベルのサイズの分布は比較的一様に近いことが伺え，質問応答時間が安定していることが期待できる．図より，大きなラベルを持つ頂点はかなり少ないという事が分かるため，もしそれらに関する応答時間が不安である場合，それらについてのみ最短距離を前もって計算しておけばよい．

7.2.3 ビットレベル並列性の活用による高速化

5 章で述べたビットレベル並列性の活用による高速化の手法の効果について議論する．図 3 はビットベクトル BFS の回数 t を変化させた時の性能の変化を表している．適切な値の設定により，索引構築時間，平均質問応答時間，索引サイズの全てが改善していることが分かる．また，t をよほど大きな値に設定しない限り性能が劣化するようなことは起こらないことが読み取れ，t の設定は比較的簡単であることが分かる．

8. おわりに

本論文では最短経路クエリ問題に対する効率的な索引構築技法である枝刈りラベリングアルゴリズムを提案した．提案手法は枝刈りを伴う幅優先探索により 2-Hop ラベルを直接的に計算するという新しいアイディアに基づく．さらに，ビットレベルの並列性の活用によってより一層の効率化を行った．非常に簡潔であるにも関わらず提案手法は既存手法より遥かに高いスケーラビリティを持ち，数億辺規模のネットワークの索引を数時間で構築し，数マイクロ秒で距離を応答することができることを実験により確認した．今まで非常に多くの手法が開発されてきた有名な問題でありながら，正確な最短経路を回答できる手法であってこのような性能を持つものは今まで存在していなかった．提案手法により，大規模なグラフデータにおいても精度を犠牲にせず様々な処理が可能となることが期待される．

謝辞 本研究の一部は，日本学術振興会特別研究員奨励費 (256563) の助成によるものである．ここに記して謝意を表す．

文　献

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, pp. 230–241, 2011.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *ESA*, pp. 24–35. 2012.

[3] T. Akiba, C. Sommer, and K. Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *EDBT*, pp. 144–155, 2012.

[4] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pp. 44–54, 2006.

[5] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang. Complex networks: Structure and dynamics. *Physics reports*, 424(4-5):175–308, 2006.

[6] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Network robustness and fragility: Percolation on random graphs. *Physical Review Letters*, 85:5468–5471, 2000.

[7] W. Chen, C. Sommer, S.-H. Teng, and Y. Wang. A compact routing scheme and approximate distance oracle for power-law graphs. *TALG*, 9(1):4:1–26, 2012.

[8] J. Cheng and J. X. Yu. On-line exact shortest distance query processing. In *EDBT*, pp. 481–492, 2009.

[9] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *SODA*, pp. 937–946, 2002.

[10] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *CIKM*, pp. 499–508, 2010.

[11] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, pp. 305–316, 2007.

[12] R. Jin, N. Ruan, Y. Xiang, and V. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD*, pp. 445–456, 2012.

[13] C. Jordan. Sur les assemblages de lignes. *J. Reine Angew Math*, 70:185–190, 1869.

[14] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pp. 137–146, 2003.

[15] R. Pastor-Satorras and A. Vespignani. *Evolution and structure of the Internet: A statistical physics approach*. Cambridge University Press, 2004.

[16] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, pp. 867–876, 2009.

[17] M. Qiao, H. Cheng, L. Chang, and J. X. Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. In *ICDE*, pp. 462–473, 2012.

[18] S. A. Rahman, P. Advani, R. Schunk, R. Schrader, and D. Schomburg. Metabolic pathway analysis web service (pathway hunter tool at cubic). *Bioinformatics*, 21(7):1189–1193, 2005.

[19] S. A. Rahman and D. Schomburg. Observing local and global properties of metabolic pathways: 'load points' and 'choke points' in the metabolic networks. *Bioinformatics*, 22(14):1767–1774, 2006.

[20] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[21] L. Tang and M. Crovella. Virtual landmarks for the internet. In *SIGCOMM*, pp. 143–152, 2003.

[22] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE*, pp. 405–416, 2009.

[23] K. Tretyakov, A. Armas-Cervantes, L. García-Bañuelos, J. Vilo, and M. Dumas. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *CIKM*, pp. 1785–1794, 2011.

[24] A. Ukkonen, C. Castillo, D. Donato, and A. Gionis. Searching the wikipedia with contextual information. In *CIKM*, pp. 1351–1352, 2008.

[25] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *CIKM*, pp. 563–572, 2007.

[26] F. Wei. Tedi: efficient shortest path query answering on graphs. In *SIGMOD*, pp. 99–110, 2010.

[27] S. A. Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. *PVLDB*, 1(1):710–721, 2008.