精度保証付き Personalized PageRank の高速化

藤原　靖宏† 　中辻　真†† 　山室　健† 　塩川　浩昭† 　鬼塚　真†

† NTT ソフトウェアイノベーションセンタ　〒 180–8585 東京都武蔵野市緑町 3-9-11
†† NTT サービスエボリューション研究所　〒 230–0847 神奈川県横須賀市光の丘 1-1

あらまし　Personalize PageRank (PPR) はグラフマイニングにおいて有用な関連度の計算方法として知られている．本論文では PPR に対して（1）特定のノードの関連度と,（2）上位 K 個のノードと（3）関連度が閾値より大きいのノードを高速かつ正確に計算することを対象とする．オリジナルの PPR の手法は問い合わせ分布に対する関連度を全てのノードに対して収束するまで繰り返し計算によって求める．この手法の問題点として特定のノードや関連度の高いノードのみの関連度を計算できないということがあげられる．提案手法は特定のノードの関連度を疎行列を用いて計算し，関連度の高いノードを関連度の下限値と上限値を用いて求める．実験により提案手法は既存手法より大幅に高速化を達成できることを確認した．

キーワード　Personalized PageRank，関連度計算，高速化，アルゴリズム，精度保証

1. はじめに

グラフはデータをノードとエッジで表現するデータ構造であり様々な分野で用いられている．近年グラフのノードの関連度を用いたパーソナライズドサーチに関心が高まっているため，ノードの関連度として様々な手法が提案されている [1, 2, 3, 4, 5, 6]．パーソナライズドサーチにおいてノードの関連度はユーザごとに異なった問い合わせ分布に応じて計算される．Personalized PageRank (PPR) は問い合わせ分布に応じてノードの関連度を計算する手法として最も注目を集めているもののひとつである [7, 8]．PPR は今までグラフ理論でよく用いられてきたノード間の最短距離などと異なり，グラフの構造的な特徴に基づいて関連度が計算できる．

PPR は概念的に以下のように説明できる．問い合わせ分布に基づいた各ノードの存在確率からランダムウォークを開始し，隣接するノードにエッジの重みに比例した確率でランダムに移動する．さらにノードに到達するたびに一定の確率で問い合わせ分布に基づいた確率でノードに戻る．この操作を再帰的に繰り返した結果として各ノードにおける定常状態確率が得られるが，PPR はこの得られた定常状態確率を関連度とする方法である．PPR は様々な分野のアプリケーションに応用されている関連度であるが計算量が高いという問題がある．

1.1 問題設定

本論文では以下の 3 つの問題を対象とする．提案手法により多くのアプリケーションがその精度を失うことなく高速に処理することが可能となる．

［**問題** 1］（特定のノードに対する関連度の計算）
Given:　問い合わせノード x と問い合わせ分布 **d**.
Find:　問い合わせ分布 **d** に対するノード x の関連度.
この問題はグラフ解析に応用が可能である．例えば White らは研究者の共著関係のグラフに PPR を用いることにより，同じ大学に所属していても異なる研究分野である研究者は低い関連度を持つことを明らかにした [9].

［**問題** 2］（上位 K 個検索）
Given:　解ノードの個数 K と問い合わせ分布 **d**.
Find:　問い合わせ分布 **d** に対して高い関連度を有する K 個のノードの集合.
この問題は推薦システムに応用可能である．Liu らは PPR により上位 K 個のノードを計算することにより，既存手法である ItemRank [4] や L^+ [10] より高い精度で推薦が可能であることを示した [11].

［**問題** 3］（閾値検索）
Given:　閾値 ϵ 問い合わせ分布 **d**.
Find:　問い合わせ分布 **d** に対して ϵ より大きいの関連度を持つノードの集合.
Liben-Nowell らは閾値より大きい関連度を持つノードを計算しグラフのリンク予測を高い精度で行えることを示した [12].

1.2 提案手法の特徴

提案手法は我々の知る限り上記 3 つの問題を包括的に解の精度を犠牲にすることなく扱える初めてのものである．この特徴は過去の既存研究 [7, 8, 13, 14, 15] が持たないものである．提案手法は疎行列を用いて高速に特定のノードの関連値を求める．また提案手法は関連度の高いノードを関連度の下限値と上限値を用いて高速に検索する．提案手法の利点として以下のものがあげられる．

高速: 提案手法は上記の手法を用いることによって既存手法より大幅に高速である．

正確: 提案手法の出力結果は正確であることが理論的に保証されている．

パラメータフリー: 既存手法と異なり，提案手法自体が必要とするパラメータはない．

PPR はその有用性により多くのアプリケーションに応用可能であるが計算コストが問題であった．提案手法によって PPR を高速に処理することが可能となり，今後多くの PPR を用いたアプリケーションが開発されることが可能になる．

論文の構成は以下の通りである．2. 章で背景知識を述べる．3. 章で提案手法の詳細を述べる．4. 章で既存手法と提案手法の比較を行う．5. 章で提案手法におけるケーススタディをあげる．

表 1　主な記号の定義
Table 1　Definition of main symbols.

記号	定義
n	グラフのノード数
m	グラフのエッジ数
x	特定のノードの関連度計算における問い合わせノード
K	上位 K 個検索における解ノードの数
ϵ	閾値検索における閾値
c	ランダムウォークにおける再スタートの確率
s_u	ノード u の関連度
$\mathbf{s}$	$n\times 1$ の関連度ベクトル
$\mathbf{d}$	$n\times 1$ の問い合わせ分布ベクトル
$\mathbf{A}$	$n\times n$ グラフの隣接行列
$\mathbf{P}$	$n\times n$ ノードの置換行列
$\mathbf{Q}$	$n\times n$ QR 分解における直交行列
$\mathbf{R}$	$n\times n$ QR 分解における上三角行列
$\|\mathbf{R}\|$	行列 $\mathbf{R}$ における非零要素数

6. 章で関連研究を述べる．7. 章で本論文をまとめる．

2. 前　準　備

まずは本論文で用いる記号を定義し，必要となる背景知識を説明する．表 1 に主な記号とその定義を示す．PPR はグラフのノードの関連度を計算するひとつの方法である．PPR では問い合わせ分布から各ノードの存在確率を決定し，決定されたノードからランダムウォークを行う．そしてランダムウォークでノードに到達するたびに一定の確率 c で問い合わせ分布に基づき各ノードに戻る．グラフのノード数を n としエッジ数を m とする．$\mathbf{s}$ を $n\times 1$ 行列とし，s_u 要素をノード u にランダムウォークする確率を表すとする．また $\mathbf{d}$ を問い合わせ分布で決定される $n\times 1$ 行列とし，問い合わせ分布の値が 0 でないノードをシードノードとする．また $\mathbf{A}$ を列が正規化されたグラフの隣接行列とする（すなわち $A_{u,v}$ 要素はノード v からノード u へランダムウォークする確率を表す）．定常状態における各ノードにおける存在確率は以下の式を再帰的に収束するまで繰り返すことで計算することができる．

$$\mathbf{s} = (1-c)\mathbf{A}\mathbf{s} + c\mathbf{d} \tag{1}$$

PPR は定常状態における確率を関連度とする．すなわち $\mathbf{s}$ 行列の s_u 要素はノード u の分布 d に対する関連度となる．

定義から繰り返し回数を t としたとき PPR の計算コストは $O((n+m)t)$ となる．そのためグラフが大規模である場合 PPR の値を計算するのは非常に計算コストが必要となる問題がある．

3. 提 案 手 法

この章では提案手法の詳細を述べる．提案手法は正確かつ高速に特定ノードの関連度を計算し，また関連度の高いノードを求めることができる特徴がある．

3.1 概　　要

特定のノードの関連度計算　問い合わせ分布に対する関連度は繰り返し計算によって定常状態における確率を計算することで求めることができる．この方法はグラフのすべてのノードの関連度を再帰的に計算するため，特定のノードの関連度のみの計算を行うことができない．再帰的に計算を行うことを避けるために，提案手法では行列計算を用いて再帰的でない形で特定のノードの関連度を計算する．この行列計算は式 (1) から得ることができる．しかしこの手法は行列が密であれば計算コストを要するという問題がある．

提案手法は疎な行列を求めて特定のノードの関連度を高速に計算する．提案手法は事前計算の段階で隣接行列におけるノードを並び替え，並び替えを行った行列の QR 分解 [16] を計算する．そして QR 分解の結果得られた行列の逆行列を計算する．適切にノードを並び替えることによってこの逆行列は疎になる．ノードの並び替えは **NP** 完全問題を解くことで得られる．QR 分解は行列の近似手法ではないため，正確に関連度を計算することができる．

上位 K 個のノードの検索　問い合わせ分布に対する上位 K 個のノードを計算するために，オリジナルの手法では全てのノードに対して関連度を計算する必要がある．しかしこの手法は計算コストが高いという問題がある．そのため提案手法では解になり得ないノードを枝刈りし，解になる可能性があるノードに対してのみ関連度を計算する．

解になる可能性があるノードを求めるために，提案手法は各ノードに対して関連度を $O(1)$ の計算量で推定する．この手法の利点として解ノードを正確に求められることがあげられる．正確な解ノードは関連度の上限値を用いることによって得られる．このため提案手法は関連度の低いノードを推定値によって高速に枝刈りすることができる．

閾値より大きいのノードの検索　関連度の上限値を用いることによって解になり得ないノードを枝刈りできるが，解の正確性を保証するために解になる可能性があるすべてのノードに対して正確な関連度を計算する必要がある．高速に閾値 ϵ より大きいのノードの検索するため，提案手法はこの正確な関連度の計算のコストを減らすために関連度の下限値を計算する．

もしあるノードの関連度の下限値が ϵ より大きければ，そのノードの正確な関連度は ϵ より大きくなる．すなわちこのノードは解ノードとなる．提案手法はこの性質を用いて正確な関連度の計算のコストを減らす．結果として提案手法は閾値より大きなノードを高速に求めることができる．

3.2 特定のノードの関連度計算

この章では問い合わせ分布に対し特定のノードの関連度を高速に計算する手法について述べる．まず 3.2.1 章で PPR の定義から QR 分解を用いることによって関連度を繰り返し計算を行わずに求められることを示す．そして 3.2.2 章で QR 分解の結果得られる行列を疎にすることは **NP** 完全問題であることを述べ，それに対する解決方法を示す．

3.2.1　繰り返しなしの関連度計算

式 (1) を変形して繰り返し計算なしで関連度が求められることを示す．提案手法ではグラフのノードを並び替える．すなわち行列 $\mathbf{P}$ をノードの置換行列 [16] とすると，グラフの隣接行列 $\mathbf{A}$ は $\mathbf{A}' = \mathbf{P}\mathbf{A}\mathbf{P}^T$ を計算し変換される．ここで行列 $\mathbf{P}^T$ は行列 $\mathbf{P}$ の転置行列である．またここで $n\times n$ の大きさの置換行列 $\mathbf{P}$ は直交行列で，各行と列においてただ一つの値が 1 である要素と，それ以外は値が 0 である要素を持つ．置換行列において $P_{ij}=1$ は j 番目の行が i 番目の行に置換されることを表す．置換行列 $\mathbf{P}$ を求める手法は 3.2.1 章で述べる．

ここで $\mathbf{I} = \mathbf{P}^T\mathbf{I}\mathbf{P}$，$\mathbf{P}^{-1} = \mathbf{P}^T$，$\mathbf{A} = \mathbf{P}^T\mathbf{A}'\mathbf{P}$ であるため（$\mathbf{P}^{-1}$ は $\mathbf{P}$ の逆行列），式 (1) は $\mathbf{P}$ を用いて以下のように書き換えられる．

$$\begin{aligned}\mathbf{s} &= c\{\mathbf{I}-(1-c)\mathbf{A}\}^{-1}\mathbf{d} = c\{\mathbf{P}^T\mathbf{I}\mathbf{P}-(1-c)\mathbf{P}^T\mathbf{A}'\mathbf{P}\}^{-1}\mathbf{d}\\ &= c\mathbf{P}^T\{\mathbf{I}-(1-c)\mathbf{A}'\}^{-1}\mathbf{P}\mathbf{d}\end{aligned} \tag{2}$$

特定のノードの関連度を計算するため，提案手法は行列 $\mathbf{I}-(1-c)\mathbf{A}'$ の QR 分解を計算する．すなわち $\mathbf{QR}=\mathbf{I}-(1-c)\mathbf{A}'$ となる．ここで行列 $\mathbf{Q}$ は直交行列であり，行列 $\mathbf{R}$ は上三角行列である．提案手法は $\mathbf{Q}^T$ と $\mathbf{R}^{-1}$ を用いて特定のノード x の関連度を以下のように計算する．

[**定義 1**]（関連度の計算） $n\times n$ の行列を $\mathbf{F}=c\mathbf{P}^T\mathbf{R}^{-1}$ とし，$n\times 1$ のベクトルを $\mathbf{g}=\mathbf{Q}^T\mathbf{Pd}$ とし，$1\times n$ のベクトルを $\mathbf{f}_i$ を行列 $\mathbf{F}$ における i 番目の行ベクトルとする．ノード x の関連度 s_x を以下のように計算する．

$$s_x=\mathbf{f}_x\cdot\mathbf{g} \tag{3}$$

定義 1 に対して以下の定理が成り立つ．

[**定理 1**]（関連度の計算） 式 (3) による関連度は式 (1) による関連度と等しくなる．

証明 $\mathbf{Q}^T=\mathbf{Q}^{-1}$ であるため [16]，式 (2) から以下の式が成り立つ．

$$\mathbf{s}=c\mathbf{P}^T(\mathbf{QR})^{-1}\mathbf{Pd}=c\mathbf{P}^T\mathbf{R}^{-1}\mathbf{Q}^T\mathbf{Pd}=\mathbf{F}\cdot\mathbf{g}$$

そのため $n\times 1$ の列ベクトル $\mathbf{s}$ の x 番目の要素 s_x（ノード x の関連度）はベクトル $\mathbf{f}_x$ とベクトル $\mathbf{g}$ の内積となる． □

式 (3) からノード x の関連度はベクトル $\mathbf{f}_x$ と $\mathbf{g}$ を用いれば繰り返し計算なしに求めることができる．またベクトル $\mathbf{d}$ はユーザから与えられるため，関連度を計算するためまずベクトル $\mathbf{g}=\mathbf{Q}^T\mathbf{Pd}$ を計算する．

関連度の計算量を述べるために以下の補助定理を示す．

[**定理 2**]（関連度の計算量） $|\mathbf{f}_x|$ と $|\mathbf{Q}|$ をそれぞれ $\mathbf{f}_x$ と $\mathbf{Q}$ における非零要素の数とすると，特定のノードの関連度を計算するために必要な計算量は $O(|\mathbf{f}_x|+|\mathbf{Q}|)$ となる．

証明 関連度を計算するためまずベクトル $\mathbf{g}$ を分布 $\mathbf{d}$ から計算する．ベクトル $\mathbf{Pd}$ は分布 $\mathbf{d}$ のノードを置換すれば得られるため，ベクトル $\mathbf{g}=\mathbf{Q}^T\mathbf{Pd}$ を計算するために必要な計算量は $O(|\mathbf{Q}|)$ となる．ベクトル $\mathbf{f}_x$ と $\mathbf{g}$ の内積を計算するための計算コストは $O(|\mathbf{f}_x|+|\mathbf{Q}|)$ であるため，関連度を計算するため必要な計算量は $O(|\mathbf{f}_x|+|\mathbf{Q}|)$ となる． □

ここで $\mathbf{F}=c\mathbf{P}^T\mathbf{R}^{-1}$ であり $\mathbf{g}=\mathbf{Q}^T\mathbf{Pd}$ であるため，定理 2 から関連度を高速に計算するためには行列 $\mathbf{R}^{-1}$ と $\mathbf{Q}$ の非零要素を減らす必要があることがわかる．なおここで明らかに行列 $\mathbf{P}$ の非零要素の数は n となる [16]．とくにベクトル $\mathbf{g}$ はユーザの問い合わせから計算されるため，行列 $\mathbf{Q}$ の非零要素の数は少なくなるようにしなければならない．次の章で行列 $\mathbf{R}^{-1}$ と $\mathbf{Q}$ を疎にする手法を述べる．

3.2.2 疎行列問題

定義 1 に示すとおり，提案手法は事前に計算した行列 $\mathbf{R}^{-1}$ と $\mathbf{Q}$ から関連度を計算する．しかしもしこれらの行列が密であれば関連度を求めるには高い計算量が必要となってしまう．これらの行列を疎にするために提案手法ではノードの並び替えを行う．しかし疎な行列を得るためにノードの並び替えを行うのは **NP** 完全問題である．

[**定理 3**]（疎行列問題） 行列 $\mathbf{R}^{-1}$ と $\mathbf{Q}$ の非零要素をノードの並び替えによって最小化するのは **NP** 完全問題である．

証明 この定理を *Minimum fill-in* 問題 [17] のリダクションから証明する．まず *Minimum fill-in* 問題のインスタンスを疎行列問題のインスタンスに置き換える．*Minimum fill-in* 問題におけるグラフを隣接行列 $\mathbf{A}$ に置き換える．またノードの削除の順番を置換行列 $\mathbf{P}$ に置き換え，弦グラフを行列 $\mathbf{R}^{-1}$ と $\mathbf{Q}$ に置き換える．このようにすれば *Minimum fill-in* 問題における最小のエッジの追加数に対する解がある時かつそのときに限り疎行列問題における非零要素の増加を最小にする解も存在することがわかる．そのため疎行列問題は **NP** 問題になる． □

本論文では疎行列問題に対する手法を示す．手法の詳細を示すためにまず行列 $\mathbf{Q}$ と $\mathbf{R}^{-1}$ がどのように計算されるかを示す．ベクトル $\mathbf{q}_i$ を行列 $\mathbf{Q}$ の i 番目の列ベクトルとし，ベクトル $\mathbf{w}_i$ を行列 $\mathbf{W}=\mathbf{I}-(1-c)\mathbf{A}'$ の i 番目の列ベクトルとする．グラム・シュミットの正規直交化法 [16] を用いることによって，行列 $\mathbf{Q}$ と $\mathbf{R}$ は以下のように計算できる．

$$\mathbf{q}_i=\mathbf{q}'_i/||\mathbf{q}'_i||,\ \mathbf{q}'_i=\begin{cases}\mathbf{w}_i & (i=1)\\ \mathbf{w}_i-\sum_{j=1}^{i-1}(\mathbf{w}_i\cdot\mathbf{q}_j)\mathbf{q}_j & (i\neq 1)\end{cases} \tag{4}$$

$$R_{ij}=\begin{cases}0 & (i>j)\\ ||\mathbf{q}'_i|| & (i=j)\\ \mathbf{w}_j\cdot\mathbf{q}_i & (i<j)\end{cases} \tag{5}$$

ここで $||\mathbf{q}'_i||$ をベクトル $\mathbf{q}'_i$ のノルム [16] とする．行列 $\mathbf{R}^{-1}$ は後退代入 [18] を用いることによって以下のように計算できる．

$$R_{ij}^{-1}=\begin{cases}0 & (i>j)\\ 1/R_{ij} & (i=j)\\ -1/R_{ii}\sum_{k=i+1}^{j}R_{ik}R_{kj}^{-1} & (i<j)\end{cases} \tag{6}$$

式 (4) と (5) から行列 $\mathbf{Q}$ と $\mathbf{R}$ の列ベクトルは左から右の列へそれぞれ直交な列ベクトルと行列 $\mathbf{W}$ との内積を計算することで求められることがわかる．式 (6) から行列 $\mathbf{R}^{-1}$ の列ベクトルは右から左の列へ，また各列ベクトルの要素は下から上の要素の順番で求められる．また式 (4) と (5) と (6) から行列 $\mathbf{Q}$ と $\mathbf{R}$ は行列 $\mathbf{W}$ から計算できることがわかる．また $\mathbf{W}$ は行列 $\mathbf{A}'(=\mathbf{PAP}^T)$ から求められる．

行列 $\mathbf{Q}$ の非零要素を減らすために，提案手法では以下の知見を用いる．「もし列ベクトル $\mathbf{w}_i$ が行列 $\mathbf{w}_i$ において全ての左の要素がすべて零である一つの非零要素をもつならば，その列ベクトル $\mathbf{w}_i$ は行列 $\mathbf{Q}$ のすべての左の列ベクトルに対して直交となる．」．これは列ベクトル $\mathbf{w}_i$ が線形独立 [16] になるためである．よって行列 $\mathbf{Q}$ は疎なデータ構造となる．また行列 $\mathbf{R}^{-1}$ を疎にするため以下の 2 つの知見を用いる．「もし行列 $\mathbf{R}$ の右と下の要素が疎であれば，行列 $\mathbf{R}^{-1}$ の右と下の要素も疎になる．」．「もし行列 $\mathbf{W}$ の右と下の要素が疎であれば，行列 $\mathbf{R}$ の右と下の要素も疎になる．」．

Algorithm 1 に疎な行列を得るためのノードの置換方法についてのアルゴリズムを示す．このアルゴリズムは行列 $\mathbf{Q}$ と $\mathbf{R}^{-1}$ に対する知見に基づいている．このアルゴリズムにおいて $\mathcal{V}$ はグラフにおけるノードの集合とし，$\mathcal{P}$ は置換されたノードの集合とする．$deg(u)$ はノード u に入るエッジの数とし，$e(u)$ はノード u から集合 $\mathcal{P}$ へ出ていないエッジの数とする．まずアルゴリズムは置換行列を零行列に初期化し，集合 $\mathcal{P}$ を空集合とする（1～2 行目）．置換されていないノードの集合 $\mathcal{V}\backslash\mathcal{P}$ からエッジの数が $\min\{e(u)|u\in\mathcal{V}\backslash\mathcal{P}\}$ になるノード集合 $\mathcal{U}$ を計算する（4 行目）．この処理は行列 $\mathbf{Q}$ に対する知見に基づいている．もし行列 $\mathbf{A}'$ の右と下の要素が疎であれば行列 $\mathbf{R}^{-1}$ も疎になるため，行列 $\mathbf{A}'$ の左と上の要素を密にす

Algorithm 1 ノードの並び替え

Input: $\mathbf{A}$, グラフの隣接行列; n, ノード数
Output: $\mathbf{P}$, 置換行列
1: $\mathbf{P} = \mathbf{0}$;
2: $\mathcal{P} = \emptyset$;
3: **for** $i = 1$ to n **do**
4: $\quad \mathcal{U} = \mathrm{argmin}(e(u)|u \in \mathcal{V} \backslash \mathcal{P})$;
5: $\quad v = \mathrm{argmax}(deg(u)|u \in \mathcal{U})$;
6: $\quad P_{iv} = 1$;
7: $\quad$ ノード v を $\mathcal{P}$ に追加;
8: **end for**
9: **return** $\mathbf{P}$;

るためにエッジの数が最大になるノードを選択する（5行目）．そして選択されたノードから行列 $\mathbf{P}$ の要素を決定し，置換の順番を決定する（6～7行目）．これらの処理を全てのノードが並び替えるまで繰り返す．

4. 章で示すとおり，この手法によって行列 $\mathbf{R}^{-1}$ と特に行列 $\mathbf{Q}$ に対して非零要素の数を減らすことができる．$\mathbf{F} = c\mathbf{P}^T\mathbf{R}^{-1}$ であり $\mathbf{g} = \mathbf{Q}^T\mathbf{P}\mathbf{d}$ であるため，提案手法は特定のノードの関連度を高速に計算可能となる．さらに非零要素の数が少なくなることにより，事前に行列 $\mathbf{Q}$ と $\mathbf{R}^{-1}$ を高速に計算可能となる．

3.3 上位 K 個のノードの検索

この章では上位 K 個のノードを検索するために，関連度の上限値を用いる手法を示す．上限値は上位 K 個のノードを検索する過程で計算する．3.3.1 章で上限値を計算するために必要となる関連度の下限値の計算方法を述べ，上限値の計算方法を 3.3.2 章で述べる．そして 3.3.3 章で検索アルゴリズムを示す．

3.3.1 関連度の下限値

関連度の上限値を計算するために提案手法はまず関連度の下限値を推定する．関連度の下限値は与えられた分布の値が 0 でないシードノードからの最小ホップ数を用いて計算する．シードノードからの最小ホップ数は幅優先探索を用いて計算する．h_u をノード u に対するシードノードからの最小ホップ数とし，$\mathcal{H}(i)$ をシードノードからの最小ホップ数が $i (i = 0, 1, \ldots)$ となるノードの集合とする．すなわち $\mathcal{H}(i)$ は i 番目のレイヤを構成し，$\mathcal{H}(0)$ はシードノードから構成される一番上のレイヤのノード集合である．関連度の下限値は以下のように計算する．

[**定義** 2]（下限値） ノード u の下限値 $\underline{s}_u$ は以下の式に基づき計算される．

$$\underline{s}_u = \begin{cases} cd_u & (u \in \mathcal{H}(0)) \\ (1-c)\sum_{v \in \mathcal{H}(h_u - 1)} A_{uv}\underline{s}_v & (u \notin \mathcal{H}(0)) \end{cases} \quad (7)$$

この式は（1）シードノードに対しては再スタートの確率と問い合わせ分布の値から下限値を計算し，（2）シードノードでなければ一つ上のレイヤのノードの下限値と遷移確率から計算することがわかる．全てのノードの下限値は $O(n+m)$ の計算コストで求めることができる．これは下限値がシードノードをルートとする幅優先探索で計算でき，幅優先探索は $O(n+m)$ で計算できるからである．

この下限値の性質について以下の補助定理を示す．

[**補助定理** 1]（下限値） $\mathcal{H}(i)$ に含まれる全てのノードに対して $\underline{s}_u \leqq s_u$ が成り立つ．

証明 証明には数学的帰納法を用いる．まず $\mathcal{H}(0)$ に含まれるノードに対して成り立つことを示す．PPR においてランダムウォークがシードノードに至るのは（1）ランダムウォークが確率 c でシードノードにジャンプするか，（2）ランダムウォークがその過程でシードノードに至るかの 2 つの場合である．（1）の場合ランダムウォークはあるシードノード u に確率 cd_u でジャンプする．そのためシードノード u の定常状態における確率は cd_u より小さくなることはない．よって $\mathcal{H}(0)$ に含まれるノードに対して成り立つ．

次に $\mathcal{H}(i-1)$ 含まれるノードに対して $\underline{s}_v \leqq s_v$ が成り立つ場合，$\mathcal{H}(i)$ 含まれるノードに対して $\underline{s}_v \leqq s_v$ が成り立つことを示す．$\mathcal{H}(i-1)$ 含まれるノードはグラフ全体のノードの部分集合であり $c, d_u \geqq 0$ であるため，式 (1) において以下が成り立つ．

$$s_u = (1-c)\sum_{v \in \mathcal{V}} A_{uv}s_v + cd_u \geqq (1-c)\sum_{v \in \mathcal{H}(i-1)} A_{uv}\underline{s}_v = \underline{s}_u$$

よって成り立つ． □

よって全てのノードに対して定義 2 から下限値を計算できる．

3.3.2 関連度の上限値

この章では不要な計算を枝刈りするために関連度の上限値を計算する手法を述べる．上限値を計算するために 3.3.1 で導入した下限値を用いる．提案手法は上位 K 個のノードを検索するために一つ一つノードをたどり上限値と正確な関連度を計算する．u_i を $i (i = 1, 2, \ldots)$ 番目にたどるノードとしたときに関連度の上限値は以下のように計算する．

[**定義** 3]（上限値） ノード u_i の上限値 $\overline{s}_{u_i}$ は以下のように定義される．

$$\overline{s}_{u_i} = \begin{cases} 1 - \sum_{j=2}^{n} \underline{s}_{u_j} & (i = 1) \\ \underline{s}_{u_i} - s_{u_{i-1}} + \overline{s}_{u_{i-1}} & (i \neq 1) \end{cases} \quad (8)$$

上限値を計算する計算コストはもし $i = 1$ であれば $O(n)$ になる．またもしそうでなければ上限値はすでに計算した $\underline{s}_{u_i}$ と $s_{u_{i-1}}$ と $\overline{s}_{u_{i-1}}$ を用いて逐次的に計算できるため，その計算コストは $O(1)$ になる．

提案手法はノードは関連度の下限値の降順でたどる．これには 2 つの理由がある．はじめの理由は下限値が大きいほど正確な下限値も大きくなることが期待されるからである．そのため効率的に上位 K 個のノードを検索できる．2 つめの理由は以下に示す補助定理により検索の途中で処理を停止し，より高速に検索を行うためである．

[**補助定理** 2]（上限値） ノードを関連度の下限値の降順にたどった場合，$s_{u_i} \leqq \overline{s}_{u_i} \leqq \overline{s}_{u_{i-1}}$ が成り立つ．

証明 まず $s_u \leqq \overline{s}_u$ になることを示す．式 (8) と補助定理 1 から以下の式が成り立つ．

$$\begin{aligned} \overline{s}_u &= \underline{s}_{u_i} - s_{u_{i-1}} + \overline{s}_{u_{i-1}} = \underline{s}_{u_i} - s_{u_{i-1}} + \underline{s}_{u_{i-1}} - s_{u_{i-2}} + \ldots + \overline{s}_{u_1} \\ &= \sum_{j=2}^{i} \underline{s}_{u_j} - \sum_{j=1}^{i-1} s_{u_j} + 1 - \sum_{j=2}^{n} \underline{s}_{u_j} = 1 - \sum_{j=1}^{i-1} s_{u_j} - \sum_{j=i+1}^{n} \underline{s}_{u_j} \\ &\geqq 1 - \sum_{j=1}^{i-1} s_{u_j} - \sum_{j=i+1}^{n} s_{u_j} = 1 - \sum_{j \neq i}^{n} s_{u_j} = s_{u_i} \end{aligned}$$

つぎに $\overline{s}_{u_i} \leqq \overline{s}_{u_{i-1}}$ が成り立つことを示す．式 (8) から $\overline{s}_{u_{i-1}} - \overline{s}_{u_i} = s_{u_{i-1}} - \underline{s}_{u_i} \geqq \underline{s}_{u_{i-1}} - \underline{s}_{u_i} \geqq 0$ となる．よって成り立つ． □

補助定理 2 からもしあるノードの上限値が解の候補ノードの K 番目の下限値より小さくなれば，他のまだたどっていないノードの関連度も K 番目の下限値より小さくなることがわかる．そのためこの場合，提案手法はその処理を打ち切る．

Algorithm 2 上位 K 個のノードの検索

Input: $\mathbf{d}$, 問い合わせ分布; K, 解ノードの個数
Output: $\mathcal{V}_a$, 解ノードの集合

1: $\theta = 0$;
2: $\mathcal{V}_e = \emptyset$;
3: $\mathcal{V}_a = \emptyset$;
4: K 個のダミーノードを $\mathcal{V}_a$ に追加;
5: すべてのノードに対して関連度の下限値を計算;
6: **while** $\mathcal{V}_e \neq \mathcal{V}$ **do**
7: $u = \mathrm{argmax}(\underline{s}_v | v \in \mathcal{V} \backslash \mathcal{V}_e)$;
8: ノード u の関連度の上限値を計算;
9: **if** $\overline{s}_u < \theta$ **then**
10: **return** $\mathcal{V}_a$;
11: **else**
12: 問い合わせ分布 $\mathbf{d}$ に対するノード u の正確な関連度を計算;
13: **if** $s_u > \theta$ **then**
14: $v = \mathrm{argmin}(s_w | w \in \mathcal{V}_a)$;
15: ノード v を集合 $\mathcal{V}_a$ から削除;
16: ノード u を集合 $\mathcal{V}_a$ に追加;
17: $\theta = \min(s_w | w \in \mathcal{V}_a)$;
18: **end if**
19: **end if**
20: ノード u を $\mathcal{V}_e$ に追加;
21: **end while**
22: **return** $\mathcal{V}_a$;

Algorithm 3 閾値より大きいノードの検索

Input: $\mathbf{d}$, 問い合わせ分布; ϵ, 検索の閾値
Output: $\mathcal{V}_a$, 解ノードの集合

1: $\mathcal{V}_e = \emptyset$;
2: $\mathcal{V}_a = \emptyset$;
3: すべてのノードに対して関連度の下限値を計算;
4: **while** $\mathcal{V}_e \neq \mathcal{V}$ **do**
5: $u = \mathrm{argmax}(\underline{s}_v | v \in \mathcal{V} \backslash \mathcal{V}_e)$;
6: ノード u の関連度の上限値を計算;
7: **if** $\overline{s}_u \leqq \epsilon$ **then**
8: **return** $\mathcal{V}_a$;
9: **else if** $\underline{s}_u > \epsilon$ **then**
10: ノード u を集合 $\mathcal{V}_a$ に追加;
11: **else**
12: 問い合わせ分布 $\mathbf{d}$ に対するノード u の正確な関連度を計算;
13: **if** $s_u > \epsilon$ **then**
14: ノード u を集合 $\mathcal{V}_a$ に追加;
15: **end if**
16: **end if**
17: ノード u を $\mathcal{V}_e$ に追加;
18: **end while**
19: **return** $\mathcal{V}_a$;

3.3.3 上位 K 個のノードを求めるアルゴリズム

Algorithm 2 に上位 K 個のノードを検索をするアルゴリズムを示す．Algorithm 2 において θ は解候補のノード集合における K 番目の関連度であり，$\mathcal{V}_a$ は解候補のノード集合であり，$\mathcal{V}_e$ は探索済みのノード集合とする．アルゴリズムは関連度が 0 である K 個のダミーノードを $\mathcal{V}_a$ に追加する（4行目）．そしてすべてのノードに対して関連度の下限値を計算する（5行目）．そして最も関連度の下限値が最も高いノードを探索していないノードの集合 $\mathcal{V} \backslash \mathcal{V}_e$ から選択し，選択されたノードの関連度の上限値を計算する（7～8行目）．補助定理 2 から選択されたノードの関連度の上限値が θ より小さい場合，そのノードおよび探索されていないノードは解ノードになり得ない．そのためその場合アルゴリズムは処理を停止する（9～10行目）．もしそうでなければ選択されたノードは解になり得る．そのためそのノードの正確な関連度を計算する（12行目）．もし計算した関連度が θ より大きければ，解候補の集合 $\mathcal{V}_a$ と K 番目の関連度 θ を更新する（13～18行目）．

このアルゴリズムが正確に解ノードを検索できることを示すために以下の定理を示す．

[**定理** 4]（正確な上位 K 個のノードの検索） Algorithm 2 により上位 K 個のノードを正確に検索できる．

証明 θ_K を解ノードにおける K 番目の関連度とした場合，解ノードの関連度の上限値は θ_K より小さくなることはない（補助定理 2）．もし関連度の上限値が θ より小さければ，アルゴリズム 2 はノードを枝刈りする．解候補のノード集合における K 番目の関連度 θ は θ_K より大きくなることはないため，解ノードがアルゴリズム 2 により枝刈りされることはない．またもしあるノードが θ より小さければ，そのノードはアルゴリズム 2 により枝刈りされる．よって成り立つ． □

提案手法の計算コストについて以下の定理を示す．

[**定理** 5]（上位 K 個のノードの検索の計算コスト） Algorithm 2 は $O(n \log n + m + |\mathbf{F}| + |\mathbf{Q}|)$ の計算コストを要する．

証明 Algorithm 2 はまずすべてのノードに対して関連度の下限値を計算するが，下限値は幅優先探索で $O(n + m)$ の計算コストで計算できる．そして関連度の下限値が最も大きいノードを選択しその関連度の上限値を計算するが，もしどのノードも枝刈りされなければこれらはそれぞれ $O(n \log n)$ と $O(n)$ の計算コストを要する．$\sum_{i=1}^{n} |\mathbf{f}_i| + |\mathbf{Q}| = |\mathbf{F}| + |\mathbf{Q}|$ であるため，定理 2 からすべてのノードの関連度を計算するには $O(|\mathbf{F}| + |\mathbf{Q}|)$ の計算コストを要する．そのため上位 K 個のノードを検索をする計算コストは $O(n \log n + m + |\mathbf{F}| + |\mathbf{Q}|)$ である． □

3.4 閾値より大きいノードの検索

この章では閾値 ϵ より大きい関連度を持つノードを検索する手法について述べる．この章では紙幅の関係から証明を省く．閾値 ϵ より大きい関連度を持つノードを検索するのに関連度の下限値を用いる．具体的にはもしあるノードの関連度の下限値が閾値 ϵ より大きい場合，明らかにそのノードの正確な関連度は閾値 ϵ より大きくなる．そのためそのようなノードに対して正確な関連度の計算を省略する．閾値 ϵ より大きい関連度を持つノードを検索するために，上位 K 個のノードを計算する場合と同様に下限値の降順にノードを探索する．そして関連度の上限値を用いて不要な関連度の計算を省略する．しかしもし正確な関連度の計算を行わない場合，定義 3 を用いて上限値を計算することができない．そのためもし正確な関連度の計算を省略した場合，以下のように関連度の上限値を計算する．

[**定義** 4]（正確な関連度を計算しない場合の上限値） もしノード u_{i-1} の正確な関連度の計算を省略した場合，ノード u_i の上限値 $\overline{s}_{u_i}$ を以下のように計算する．

$$\overline{s}_{u_i} = \underline{s}_{u_i} - \underline{s}_{u_{i-1}} + \overline{s}_{u_{i-1}} \tag{9}$$

この定義について以下の補助定理を示す．

[**補助定理** 3]（Upper bound w/o exact relevances） 定義 4 に対して i 番目に探索したノード u_i に対して $s_{u_i} \leqq \overline{s}_{u_i} \leqq \overline{s}_{u_{i-1}}$ が成り立つ．また上限値を計算する計算コストは $i \neq 1$ であれば $O(1)$ になる．

Algorithm 3 に関連度が閾値 ϵ より大きくなるノードを検索するアルゴリズムを示す．このアルゴリズムはすべてのノードに対して関連度の下限値を計算する（3行目）．そして最も関連度の下限値が最も高いノードを探索していないノードの集合から選択し，選択されたノードの関連度の上限値を計算する（5～6行目）．もしその上限値が ϵ より大きくなければ，そのノードおよび探索されていないノードは解ノードになり得ない．そのためその場合アルゴリズムは処理を停止する（7～8行目）もし選択されたノードの下限値が ϵ より大きければ，そのノードは解ノードとなる．そのためそのノードを解ノードの集合に追加する（9～10行目）．そうでなければ正確な関連

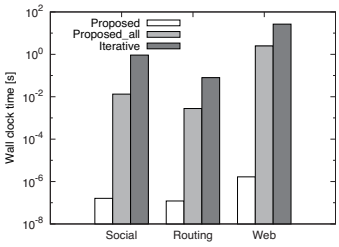

図 1　関連度の計算時間.
Fig. 1　Relevance computation time.

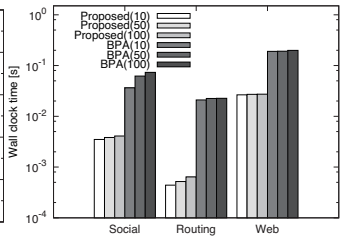

図 2　上位 K 個のノードの計算時間.
Fig. 2　Efficiency of top-k nodes identification.

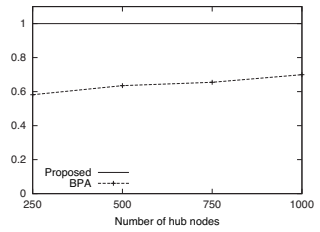

図 3　ハブノードの数と適合率.
Fig. 3　Accuracy versus number of hub nodes.

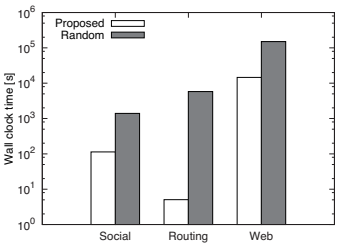

図 4　事前計算時間.
Fig. 4　Precomputation time.

表 2　非零要素の割合.
Table 2　Ratio of non-zero elements.

	Data set		
	Social	*Routing*	*Web*
$\lvert\mathbf{Q}\rvert/n^2$	3.40×10^{-6}	4.88×10^{-6}	2.76×10^{-4}
$\lvert\mathbf{R}^{-1}\rvert/n^2$	9.25×10^{-3}	2.40×10^{-3}	1.93×10^{-2}

度を計算し，選択されたノードが解ノードかの確認を行う（１１～１６行目）.

Algorithm 3 の性質について以下の定理を示す.

［**定理** 6］（閾値より大きいノードの検索）　Algorithm 3 は関連度が閾値 ϵ より大きくなるノードを正確に $O(n \log n + m + |\mathbf{F}| + |\mathbf{Q}|)$ の計算コストで行う.

4. 評 価 実 験

提案手法の有効性を示すために評価実験を行った．実験では以下のデータセットを用いた.

Social(注1): Slashdot.org(注2) におけるユーザ間の関係グラフである．ノード数は 82,168 でエッジ数は 948,464 である.

Routing(注3): Oregon Route Views Project(注4) により得られたネットワークのルーティングのデータである．ノード数は 22,963 でエッジ数は 48,436 である.

Web(注5): ノートルダム大学のサイトから得られたグラフである．ノード数は 325,729 でエッジ数は 1,497,135 である.

PPR における再スタートの確率は関連研究 [2] と同様に $c = 0.9$ とした．実験ではランダムに 10 個のシードノードを選択し，それらの問い合わせ分布の大きさは等しい値とした．実験は CPU が Intel Xeon 2.26GHz，メモリが 144 GB の Linux サーバで行った．すべてのアルゴリズムは GCC で実装した.

4.1　関連度の計算時間

提案手法で特定のノードの関連度を計算するために必要な処理時間についてオリジナルの手法と比較実験を行った．オリジナルの手法は特定のノードの関連度を計算できないため，提案手法で全てのノードを関連度を計算するために必要な処理時間もあわせて示す.

図 1 に結果を示す．ここで提案手法で特定のノードの関連度を計算した場合の結果を「Proposed」，全てのノードの関連度を計算した場合の結果を「Proposed_all」に示す．また「Iterative」はオリジナルの繰り返し計算により関連度を計算した場合の結果である．また表 2 に行列 $\mathbf{Q}$ と $\mathbf{R}^{-1}$ における非零要素の割合を示す.

図 1 から提案手法はオリジナルの手法より 7 桁以上高速であることがわかる．また全てのノードの関連度を計算したとしても 70 倍以上高速である．定理 2 に示すとおり，提案手法は $O(|\mathbf{f}_x| + |\mathbf{Q}|)$ の計算コストで計算できる．ここで $\mathbf{F} = c\mathbf{P}^T\mathbf{R}^{-1}$ であり $\mathbf{g} = \mathbf{Q}^T\mathbf{P}\mathbf{d}$ であるため，提案手法において非零要素の数が関連度の計算時間に影響する．表 2 に示すとおり，提案手法は行列 $\mathbf{R}^{-1}$，とくに行列 $\mathbf{Q}$ を疎行列にすることができる．すなわち $|\mathbf{Q}| \ll |\mathbf{R}^{-1}| \ll n^2$ となる．そのため提案手法は与えられた問い合わせ分布 $\mathbf{d}$ に対してベクトル $\mathbf{g}$ を高速に計算できるため，関連度を高速に計算できる．一方 2. 章で述べたとおり，オリジナルの手法は全てのノードの関連度を収束するまで繰り返し計算を行うため，計算コストが高くなる．結果として提案手法はオリジナルの手法より関連度を高速に計算できる.

4.2　上位 K 個のノードの計算時間

提案手法は上位 K 個のノードを高速に検索できることを示すために Basic Push Algorithm [15] と比較を行った．この手法は上位 K 個のノードを近似的に検索する手法である．ハブノードから全てのノードに対する関連度を事前に計算して，この手法は関連度の上限値を計算する．関連度の上限値を用いるため，この手法において解ノードの再現率(注6)は 1 になる．すなわちオリジナルの手法で得られる解ノードの全てを Basic Push Algorithm によって得ることができる．他の近似手法 [13, 14] にはこのような性質はない．しかしこれは同時に Basic Push Algorithm が解ではないノードも出力することも示している．実験においてハブノードはノードの次数が高いものとした.

図 2 にハブノードの数を 1,000 個にした場合の結果を示す．この図において提案手法と Basic Push Algorithm の結果をそれぞれ「Proposed(K)」,「BPA(K)」とした．ここで K は解ノードの個数である．また図 3 に Social データに対してハブノードの数を変えたときの上位 10 個の解に対する適合率を示す．適合率は各手法で得られた解ノードのうちどの程度オリジナルの手法による解ノードと一致しているかの割合である.

図 2 にみられるように，提案手法は Basic Push Algorithm より 50 倍以上高速に上位 K 個のノードを検索できる．提案手法は全てのノードに対して上限値は $O(n)$ の計算コストで求めることができ，また上限値を用いて正確な関連度の計算を枝刈りする．そのため提案手法は高速な検索が可能である.

図 3 から提案手法はオリジナルの手法と同じ結果を出力することがわかる．また図 3 から Basic Push Algorithm はハブ

(注1)：http://snap.stanford.edu/data/soc-Slashdot0902.html
(注2)：http://slashdot.org/
(注3)：http://www-personal.umich.edu/~mejn/netdata/as-22july06.zip
(注4)：http://routeviews.org/
(注5)：http://vlado.fmf.uni-lj.si/pub/networks/data/ND/NDnets.htm

(注6)：再現率はオリジナルの手法で得られる解ノードうちどの程度正しく解ノードが得られるかの割合である.

(1) Proposed (2) Monte Carlo(250) (3) Monte Carlo(1,000,000)

図 5　各手法で得られた関連度の高い都市.

Fig. 5　Highly relevant cities as detected by each approach.

ノードの数が多くなるほど高い精度で上位 K 個のノードを検索できることがわかる．これは Basic Push Algorithm が事前に計算したハブノードからの関連度を用いて上限値を計算するからである．しかし Basic Push Algorithm は正確な解ノードを求めることができない．さらにハブノードからの関連度は繰り返し計算によって求めなければならないため，Basic Push Algorithm の事前計算時間はハブノードの数が多くなるほど長くなる．図 2 と 3 は提案手法が Basic Push Algorithm に対して計算時間と精度の両面で優れていることを示している．

4.3　事前計算時間

提案手法は関連度を計算するために事前に行列 $\mathbf{Q}$ と $\mathbf{R}^{-1}$ を計算する．そのためノードの置換による事前計算時間の影響について実験を行った．図 4 において「Random」はノードをランダムな順番に並び替えたときの事前計算時間を示す.

図 4 から提案手法におけるノードの並び替えは非零要素の数を減らすのみでなく事前計算の時間を減らすことができることがわかる．提案手法はランダムにノードを並び替えた場合と比較して 1,200 倍まで高速に事前計算を行えることがわかる．提案手法により行列 $\mathbf{Q}$ と $\mathbf{R}^{-1}$ を疎にすることができるため，式 (4) と (5) と (6) における非零要素の計算を高速行うことができる．そのため関連度の計算において行列 $\mathbf{Q}$ と $\mathbf{R}^{-1}$ を高速に計算することができる.

5.　ケーススタディ

提案手法は関連度が ϵ より大きいノードを正確に検索することができる．提案手法の有効性を示すために実データを用いたケーススタディを実施した．Basic Push Algorithm を含む過去のいずれの手法も閾値より大きいノードを検索するものがないため，この章では Avrachenkov らによる最新の PPR の近似手法 [13] と比較を行った．この手法はシードノードからランダムウォークを複数回計算する Monte Carlo 法を用いて関連度を近似する.

ケーススタディでは北アメリカにおける都市間のエアラインネットワークのデータ (注7) を用いた．このデータにおいてノードに対応する都市の数は 456 であり，エッジに対応する直行便の数は 71,959 である．エッジの重みは都市間の近接度に比例する．ケーススタディでは閾値を 5×10^{-4} に設定し，サンディエゴ，デンバー，オーランド，ニューヨークの 4 都市に対して関連度の高い都市を計算した．これらの 4 都市における問い合わせ分布は等しい値に設定し，前の章と同様に $c = 0.9$ とした.

(注7)：http://www.psi.toronto.edu/index.php?q=affinity propagation

図 5 に結果を示す．Monte Carlo 法による手法においてランダムウォークの数を 250 とした結果は図 5-(2) に，1,000,000 とした結果は図 5-(3) に示す．図 5 において丸はシードノードを表し，三角は関連度の高いノードを示す.

図 5-(1) に提案手法で求めたサンディエゴ，デンバー，オーランド，ニューヨークの 4 都市に対して関連度の高い都市を示す．これら 4 つの都市はいずれもアメリカで代表的な都市であり，大きなハブ空港を持つ．ハブ空港とはその他の多くの空港へ直行便で行くことができる空港である．PPR はシードノードからのランダムウォークを用いてノードの関連度を計算する手法であるため，もし大きな都市がシードノードとして選ばれた場合，多くの直行便で他の都市と結ばれている都市が関連度の高いノードとなることが期待される．その予想通り関連度の高い都市として大きなハブ空港を持つロサンゼルス，フェニックス，コロラドスプリングス，デイトナビーチ，フィラデルフィア，ワシントン D.C. が得られた．すなわち大きな都市をシードノードとすることでハブ空港を持つ都市を検索することができる．なお提案はオリジナルの手法と同じ結果になることが保証されている.

図 5-(2) にみられるように，ランダムウォークの数を 250 にした場合，Monte Carlo 法に基づく手法はコロラドスプリングスとワシントン D.C. 以外は提案手法と異なる結果となった．各手法の処理時間は Monte Carlo 法に基づく手法が 21 マイクロ秒であり，提案手法が 20.5 マイクロ秒であり，ほぼ同じであった．図 5-(3) に示すとおり，ランダムウォークの数が多くなれば Monte Carlo 法に基づく手法はボルチモアを除いて提案手法と同じになる．しかし Monte Carlo 法に基づく手法はランダムウォークの数が多くなると計算時間を要するという問題がある [13]．ランダムウォークの数を 1,000,000 にしたとき Monte Carlo 法に基づく手法の処理時間は 80.7 ミリ秒であり，提案手法の方が 3,900 倍以上高速であった.

6.　関 連 研 究

これまで PPR を高速に計算するため様々な手法が提案されてきた．しかし過去の手法のいずれも解の正確性を保証するものではない．すなわちこれらの手法の出力結果はオリジナルの手法と異なるものである．そのためこれらの手法でアプリケーションの質を保ちながら高速化を果たすことは難しい．また過去の手法のいずれも本論文で対象とした問題設定を包括的に解決するものはない.

Jeh らはハブノードからの関連度から関連度を近似する手法を示した [8]．この手法は問い合わせ分布に対する特定のノード

の関連度をハブノードからの関連度の線形結合として近似した．Fogaras らは Monte Carlo 法を PPR に応用した [7]．近似の関連度を求めるために，彼らはまず fingerprint とよばれるランダムウォークを複数計算し，線形結合を用いて近似の関連度を計算した．Avrachenkov らと Bahmani らは独立に PPR に対して Monte Carlo 法を改善する手法を提案した [13, 14]．Gupta らによって提案された Basic Push Algorithm は上位 K 個のノードを近似的に求める手法である [15]．この手法は関連度の上限値を事前に計算したハブノードからの関連度を用いて計算する．しかしこの手法はオリジナルの手法では解にならないノードも解として出力してしまう．Chakrabarti らは Basic Push Algorithm を ER グラフに対して適用した [19]．

Random walk with restart (RWR) [2] は一つのノードのみをシードノードとする PPR の特殊例である．Sun らは RWR に対する近似手法を提案した [20]．彼らの手法は METIS [21] を用いてグラフをサブグラフに分割し，シードノードを含むサブグラフに対してのみ RWR を計算する．すなわちサブグラフに含まれないノードの関連度は 0 にする．彼らの手法はシードノードから近いノードのみが関連度が高くなるという性質を用いている．同じ性質を用いて K-dash は上位 K 個のノードを高速に検索する手法である [22]．関連度の上限値を用いて K-dash は正確かつ高速に検索が可能であるが，この手法は PPR の様に複数のノードがシードノードとなる場合は処理できず，また本論文で対象とした問題を包括的に処理するものでもない．Tong らは固有値分解 [16] を用いて RWR を近似する手法を提案した [23]．彼らは彼らの手法が保証された近似誤差を持つことを示したが，この近似誤差は normalized graph Laplacian [24] をグラフを表現する形式にのみ成立するという制約がある．

7. ま と め

本論文では PPR に対して特定のノードの関連度と関連度の高いノードを高速かつ正確に求める問題に取り組んだ．提案手法は特定ノードの関連度を疎な行列を用いて繰り返し計算を用いずに求め，関連度の下限値と上限値を用いてノードを枝刈りすることで関連度の高いノードを求める．実験により提案手法が既存の手法より大幅に高速に処理が可能であることを示した．PPR は多くのアプリケーションに利用されるが，提案手法を用いることによって今後多くのアプリケーションに対して精度を犠牲にすることなく高速化が可能になることが期待される．

文　献

[1] A. Balmin, V. Hristidis and Y. Papakonstantinou: “ObjectRank: Authority-based Keyword Search in Databases”, VLDB, pp. 564–575 (2004).

[2] B. Gallagher, H. Tong, T. Eliassi-Rad and C. Faloutsos: “Using Ghost Edges for Classification in Sparsely Labeled Networks”, KDD, pp. 256–264 (2008).

[3] B. Gao, T.-Y. Liu, W. Wei, T. Wang and H. Li: “Semi-supervised Ranking on Very Large Graphs with Rich Metadata”, KDD, pp. 96–104 (2011).

[4] M. Gori and A. Pucci: “ItemRank: A Random-walk Based Scoring Algorithm for Recommender Engines”, IJCAI, pp. 2766–2771 (2007).

[5] V. Hristidis, L. Raschid and Y. Wu: “Scalable Link-based Personalization for Ranking in Entity-relationship Graphs”, WebDB (2011).

[6] G. Jeh and J. Widom: “SimRank: a Measure of Structural-context Similarity”, KDD, pp. 538–543 (2002).

[7] D. Fogaras, B. Rácz, K. Csalogány and T. Sarlós: “Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments”, Internet Mathematics, **2**, 3 (2005).

[8] G. Jeh and J. Widom: “Scaling Personalized Web Search”, WWW, pp. 271–279 (2003).

[9] S. White and P. Smyth: “Algorithms for Estimating Relative Importance in Networks”, KDD, pp. 266–275 (2003).

[10] F. Fouss, A. Pirotte, J.-M. Renders and M. Saerens: “Random-walk Computation of Similarities between Nodes of a Graph with Application to Collaborative recommendation”, IEEE Trans. Knowl. Data Eng., **19**, 3, pp. 355–369 (2007).

[11] Q. Liu, E. Chen, H. Xiong and C. H. Q. Ding: “Exploiting User Interests for Collaborative Filtering: Interests Expansion via Personalized Ranking”, CIKM, pp. 1697–1700 (2010).

[12] D. Liben-Nowell and J. M. Kleinberg: “The Link Prediction Problem for Social Networks”, CIKM, pp. 556–559 (2003).

[13] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova and M. Sokol: “Quick Detection of Top-k Personalized PageRank lists”, WAW, pp. 50–61 (2011).

[14] B. Bahmani, A. Chowdhury and A. Goel: “Fast Incremental and Personalized PageRank”, PVLDB, **4**, 3, pp. 173–184 (2010).

[15] M. S. Gupta, A. Pathak and S. Chakrabarti: “Fast Algorithms for Top-k Personalized PageRank Queries”, WWW, pp. 1225–1226 (2008).

[16] D. A. Harville: “Matrix Algebra From a Statistician’s Perspective”, Springer (2008).

[17] M. Yannakakis: “Computing the Minimum fill-in is NP-complete”, SIAM. J. on Algebraic and Discrete Methods, **2**, 1, pp. 77–79 (1981).

[18] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery: “Numerical Recipes 3rd Edition”, Cambridge University Press (2007).

[19] S. Chakrabarti, A. Pathak and M. Gupta: “Index Design and Query Processing for Graph Conductance Search”, VLDB J., **20**, 3, pp. 445–470 (2011).

[20] J. Sun, H. Qu, D. Chakrabarti and C. Faloutsos: “Neighborhood Formation and Anomaly Detection in Bipartite Graphs”, ICDM, pp. 418–425 (2005).

[21] G. Karypis, V. Kumar and V. Kumar: “Multilevel K-way Partitioning Scheme for Irregular Graphs”, Journal of Parallel and Distributed Computing, **48**, pp. 96–129 (1998).

[22] Y. Fujiwara, M. Nakatsuji, M. Onizuka and M. Kitsuregawa: “Fast and Exact Top-k Search for Random Walk with Restart”, PVLDB, **5**, 5, pp. 442–453 (2012).

[23] H. Tong, C. Faloutsos and J.-Y. Pan: “Fast Random Walk with Restart and Its Applications”, ICDM, pp. 613–622 (2006).

[24] D. Zhou, O. Bousquet, T. N. Lal, J. Weston and B. Schölkopf: “Learning with Local and Global Consistency”, NIPS (2003).