選択的重要度先読みを用いた ObjectRank の高速化

佐藤　朋紀†　　塩川　浩昭††　　北川　博之††

† 筑波大学大学院システム情報工学研究科　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1–1–1
E-mail: †t.sato@kde.cs.tsukuba.ac.jp, ††{shiokawa,kitagawa}@cs.tsukuba.ac.jp

あらまし　ObjectRank は，データベース内のオブジェクトをグラフのノードとみなすことでキーワード検索を実現するグラフ分析手法である．しかしながら，ObjectRank は重要度評価の際にノード数に等しいサイズの行列ベクトル積を繰り返し計算する必要があるため，大規模なグラフへの適用が難しいという問題がある．そこで本稿では，ObjectRank の top-k 検索の高速化手法を提案する．提案手法は，繰り返し計算の過程で各ノードの重要度の上限値と下限値を計算することで，逐次的に解になり得ないノードをグラフから枝刈りする．また，提案手法は一部のノードの重要度の上限値を任意の回数だけ先読みすることにより，枝刈りのさらなる高速化を図る．本稿では実データを用いた評価実験を行い，ObjectRank に対する提案手法の有効性を検証する．

キーワード　ObjectRank，グラフマイニング

1. 序　　論

ソーシャルメディアやモバイル機器の普及に伴い日々大量のデータが生成されており，データから新たに知識を獲得するデータ分析の需要が増加している．特に，データ間の関係性を表現するグラフと呼ばれるデータ構造に着目したグラフ分析は，データ間の関係性に隠れた知識を獲得できる分析技術であり，推薦システムや SNS(Social Networking Service) といったサービスで幅広く用いられている [1], [9], [15].

グラフ分析技術のひとつに，ObjectRank [2], [10] がある．ObjectRank は，PageRank [3] を拡張することでデータベース内のオブジェクトに対するキーワード検索を実現する手法である．データベース内のオブジェクトをグラフに見立てることによってランダムウォークに基づくリンク解析手法を適用し，各オブジェクトの重要度を評価しランク付けする．PageRank とは異なり，ObjectRank は複数の種類のノードとエッジからなるグラフを扱うことができるため，多様なデータに対して適用可能である [5], [8], [11], [12], [16].

幅広いアプリケーションが存在する一方で，ObjectRank はクエリが与えられると行列ベクトル積の繰り返し演算によりグラフ全体のノードの重要度を評価する必要があるため，計算コストが膨大であるという問題がある．グラフ内のノード数を N，エッジ数を M，演算の繰り返し回数を T とすると，重要度評価の計算量は $O((N+M)T)$ となるため，ノード数に比例して計算時間は増大する．一般的に，ObjectRank は応答時間の高速化のために，事前にデータベース内に現れる全てのキーワードに対して各ノードの重要度を評価し索引付けする．しかし，データベース内のキーワード数を K とすると索引構築の計算量は $O((N+M)TK)$ となり，大規模なグラフを対象としたとき K，N，M が爆発的に増加し，索引構築に膨大な時間を必要とする．したがって，ObjectRank の高速化は重要な研究課題となっている．

ObjectRank の重要度評価の高速化手法として，様々な手法が提案されてきた [4], [13], [14]．これらの手法は，グラフから一部のノードを取り除くことで計算対象のノード数を削減したり，重要度評価の効率的な計算のために索引を事前計算することで高速な計算を実現した．しかしながら，依然として (1) 事前計算に膨大な時間を要する，(2) 大規模なグラフを対象とした場合に計算時間がかかってしまうといった問題が存在する．

1.1 本研究の貢献

本研究では，ObjectRank の Top-k 検索を高速化する手法を提案する．提案手法では，繰り返し計算の各過程で従来の ObjectRank により得られる重要度の上限値と下限値を推定し，Top-k 検索の結果になり得ないノードをグラフから逐次的に枝刈りする．この手法により重要度評価の繰り返し計算を進めるにつれて計算対象のノードが減少するため，従来の ObjectRank と比較して Top-k 検索を高速に行うことができる．提案手法はグラフからノードの枝刈りを行うが，上限値と下限値は繰り返し計算を進めるにつれて従来の ObjectRank により得られる重要度に収束するため，得られる上位 k 件のノードは従来の ObjectRank により得られる結果と一致することが保証される．

また，我々が行った予備実験により，提案手法は繰り返し計算の序盤ではグラフに含まれるノード全体と比較して約 1 % のノードに対してのみ枝刈りが行われ，繰り返し計算の終盤の数回で残りの大部分のノードが枝刈りされることが明らかとなった．そこで提案手法は，一部のノードの上限値をユーザが指定した任意の回数だけ先読みすることで，繰り返し計算の早い段階においても枝刈りが行われることを可能にし，枝刈りのさらなる高速化を目指す．

本研究の貢献は下記の通りである．

- **高速性:** 提案手法は従来のべき乗法を用いた ObjectRank と比較して上位 k 件の結果を高速に計算することができる．

表 1　本稿が用いる記号の定義

記号	定義
G	計算対象のグラフ
V	G に含まれるノードの集合
E	G に含まれるエッジの集合
N	グラフのノード数
M	グラフのエッジ数
K	データベース内のキーワード数
$\boldsymbol{A}$	$N \times N$ の遷移行列
d	ダンピングファクタ
$\boldsymbol{r}_t$	キーワード t に対する各ノードの重要度を並べた $N \times 1$ のベクトル
$\boldsymbol{q}_t$	キーワード t についてのクエリベクトル
$BS(t)$	キーワード t をもつノードの集合

提案手法は 100 万のノードと 500 万のエッジを持つグラフに対し，約 2 秒で検索結果を得ることができる（4.1 節）．

- **正確性:** 提案手法は ObjectRank の上位 k 件の結果を高い近似精度で導出することができる．上限値の先読みを行わない場合は正確な結果が得られる（4.2 節）．

上述の通り，ObjectRank は行列ベクトルの繰り返し演算を行うため計算コストが膨大であり，大規模グラフに適用することは難しい．しかし，提案手法は逐次的なノードの枝刈りを行うことで計算対象のノード数を削減し，上位 k 件の結果を高速に得ることができる．実データを用いた評価実験では，従来のべき乗法を用いた ObjectRank と比較して約 5 倍高速に計算できることを示した．

本稿の構成は次の通りである．まず，2. 節で前提となる知識について説明し，3. 節で提案手法について述べる．4. 節で評価実験について説明し，5. 節で関連研究を紹介する．そして最後に 6. 節でまとめを述べる．

2. 前提知識：ObjectRank

表 1 に本稿が用いる記号とその定義を示す．本節では，2.1 節で ObjectRank の概要について，2.2 節で ObjectRank の重要度評価について，2.3 節で ObjectRank の問題点についてそれぞれ述べる．

2.1 ObjectRank の概要

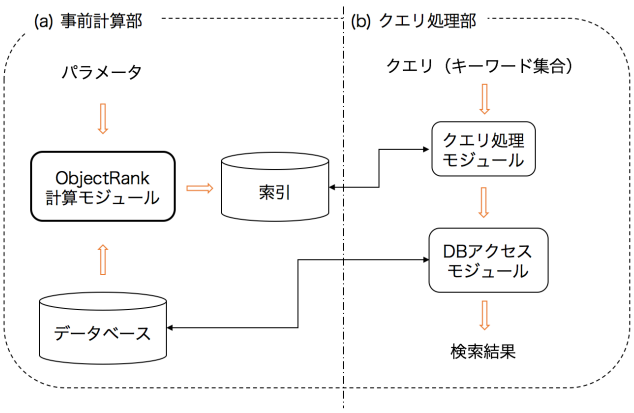

図 1　ObjectRank のシステム

ObjectRank のシステムの概要を図 1 に示す．ObjectRank のシステムは (a) 事前計算部と (b) クエリ処理部の二つに分けることができる．(a) 事前計算部では，各種パラメータを引数にとり ObjectRank 計算モジュールにおいてデータベース内の各オブジェクトの重要度を計算し，結果を索引付けする．(b) クエリ処理部ではユーザから与えられたクエリと索引に基づいてオブジェクトのランキングを作成し，重要度が上位 k 件のオブジェクトをユーザに返す．以降では ObjectRank の事前計算部についてより詳細な説明を行う．

2.2 ObjectRank の重要度評価

2.2.1 データモデル

ObjectRank はデータベース内のオブジェクトをラベル付き有向グラフとして表現する．まず，グラフのノードとエッジの種類，及びエッジの重みを定義した *Authority Transfer Schema Graph* を作成する（以降，*Schema Graph* と呼ぶ）．各種ノードとエッジはラベルが付与され，ラベルによって種類が区別される．各エッジの重みはそのエッジによって遷移する重要度の割合を示す．ただし，重みは 0 以上 1 以下の値をとり，一つのノードから出るエッジの重みの総和は 1 以下に設定する必要がある．図 2 に文献データベースを表現した *Schema Graph* の例を示す．図 2 の例では，“Conference” や “Year” といったラベルが付与された 4 種類のノードと，それらをつなぐ 8 種類のエッジが存在する．

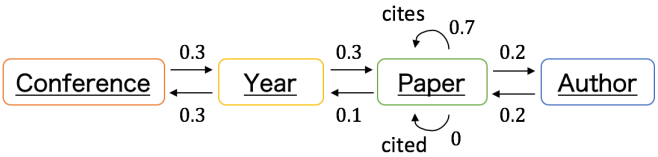

図 2　文献データベースの *Schema Graph*

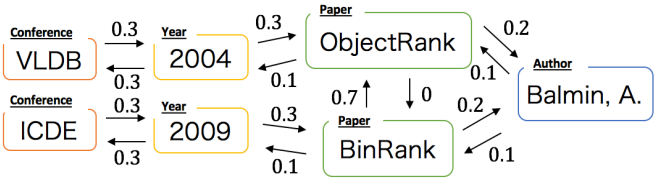

図 3　図 2 に基づいて作成された *Data Graph* の例

次に，*Schema Graph* に基づいて対象の全データを実体化したグラフ *Authority Transfer Data Graph* を構築する（以降，*Data Graph* と呼ぶ）．*Data Graph* の各ノードはデータベース内の 1 つのオブジェクトに対応し，オブジェクトに含まれるキーワード集合を保持する．*Data Graph* の各エッジの重みは，*Schema Graph* で定義した重みをエッジの元ノードの出次数で割った値となる．ただし，出自数は同じ種類のエッジの本数のみを数えたものとする．図 3 に 図 2 に基づいて作成された文献データベースの *Data Graph* の例を示す．図 3 では，“Balmin, A.” ノードから “ObjectRank” ノードには重みが 0.1 であるエッジが張られている．これは，*Schema Graph* の “Author” から “Paper” へのエッジの重みである 0.2 を，“Balmin, A.” の出次数 2 で割った値である．

2.2.2 重要度の計算方法

ObjectRank は，上述の方法により構築された *Data Graph* に対して重要度評価を行う．ノードとエッジの集合をそれぞれ V，E，ノード数とエッジ数をそれぞれ N，M とする．あるキーワード t と *Data Graph* $G=(V,E)$ が与えられたとき，ObjectRank はキーワード t に対する各ノードの重要度を並べたベクトル $\boldsymbol{r}_t=[r_t(v_1),r_t(v_2),\ldots,r_t(v_N)]$ をランダムウォークモデル [3] に基づいて計算する．すなわち，キーワード t を持つノード集合を $BS(t)$ とすると，ランダムサーファは $BS(t)$ に含まれるランダムなノードを始点として (1) エッジの重みに応じた確率で隣接ノードへと遷移，または (2) $BS(t)$ のランダムなノードへとジャンプする行動を繰り返す．ある時点でランダムサーファが滞在している確率を，そのノードの重要度と見なす．これらの処理は以下の式 (1) で定式化することができる．

$$\boldsymbol{r}_t = d\boldsymbol{A}\boldsymbol{r}_t + (1-d)\boldsymbol{q}_t \tag{1}$$

ただし，$\boldsymbol{A}$ は $N\times N$ の遷移行列であり，(i,j) 要素にはノード v_j から v_i にエッジ e_{ji} が存在する場合は e_{ji} の重みを，それ以外の場合は 0 を格納する．d $(0<d<1)$ はダンピングファクタ，$\boldsymbol{q}_t$ は n 次元のクエリベクトルであり，以下のように求める．

$$q_t(v)=\begin{cases}1/|BS(t)| & (v\in BS(t))\\ 0 & (otherwise)\end{cases}$$

ObjectRank は，べき乗法を用いて式 (1) を解く．すなわち，任意の初期値を与えて以下の式を繰り返し計算する．

$$\boldsymbol{r}_t^{(i+1)} \leftarrow d\boldsymbol{A}\boldsymbol{r}_t^{(i)} + (1-d)\boldsymbol{q}_t \tag{2}$$

$\boldsymbol{r}_t^{(i)}$ は i 回目の繰り返しによって求めた重要度のベクトルである．$\parallel \boldsymbol{r}_t^{(i+1)}-\boldsymbol{r}_t^{(i)} \parallel$ が十分小さくなったとき，重要度が収束したとして繰り返し計算を終了する．

2.2.3 索 引 構 築

ObjectRank はクエリ応答の効率化のために，クエリ処理時に重要度評価を行う代わりに事前に索引構築を行う．まずデータベースに現れるキーワード t について各ノードの重要度 $\boldsymbol{r}_t$ を計算し，ノードの id $id(u)$ と重要度 $r_t(u)$ のペア $<id(u),r_t(u)>$ を $r_t(u)$ の降順で並べたリストを作成する．次に，重要度 $r_t(u)$ が任意の閾値を下回るノードのペアをリストから取り除き，リストを索引に格納する．最後に，上述の処理をデータベースに現れる全てのキーワードについて実行することで索引を構築する．

2.3 ObjectRank の問題点

ObjectRank は，べき乗法を用いて各ノードの重要度が収束するまで式 (2) を繰り返し計算する必要があるため，重要度評価の計算量は $O((N+M)T)$ となる．索引構築時にはこの計算をデータベース内に現れる全てのキーワードに対して行う必要があり，索引構築の計算量は $O((N+M)TK)$ となる．大規模なグラフを対象とした際には N，M，および K が増加するため，計算コストが爆発的に増大するという問題がある．

3. 提 案 手 法

3.1 提案手法の概要

提案手法では，繰り返し計算の各過程で従来の ObjectRank により得られる重要度の収束値の上限値と下限値を推定し，上位 k 件になり得ないノードを発見しグラフから逐次的に枝刈りする．この手法により繰り返し計算を進めるにつれてノードが減少し，k 件の検索結果が得られた時点で計算を終えることができるため，従来の ObjectRank と比較して Top-k 検索を高速に行うことができる．理論的な解析により，i 番目の繰り返し計算における上限値 $\overline{r}_t^{(i)}(v)$ と下限値 $\underline{r}_t^{(i)}(v)$ は重要度の収束値 $\boldsymbol{r}_t(v)$ に対し $\underline{r}_t^{(i)}(v) \leqq r_t(v) \leqq \overline{r}_t^{(i)}(v)$ の性質を満たし，得られる上位 k 件の検索結果は従来の ObjectRank により得られる結果と一致することが保証される．

また我々が行った予備実験により，上述の提案手法は繰り返し計算の序盤では全体のノードと比較して約 1 % のノードに対してのみ枝刈りが行われ，繰り返し計算の終盤において残りの大部分のノードが枝刈りされることが明らかとなった．そこで提案手法は，i 番目の繰り返し計算時に一部のノードの上限値を任意の数 s だけ先読みし，$\overline{r}_t^{(i+s)}(v)$ を推定する．そして，得られた推定値を用いることで繰り返し計算の序盤においてもノードの枝刈りを可能とし，枝刈りのさらなる高速化を図る．

以降では， 3.2 節において上限値と下限値の定義とその性質の議論を行い， 3.3 節で提案手法の基本的なアルゴリズムを説明する．そして 3.4 節において上限値の先読みを導入し， 3.5 節で先読みを用いた場合のアルゴリズムを説明する．

3.2 上限値と下限値の導出

$\boldsymbol{p}_t^{(i)}$ を長さ i のランダムウォークの確率を表す N 次元ベクトルとする．$\boldsymbol{p}_t^{(i)}=\boldsymbol{A}^i\boldsymbol{q}_t$ で計算し，$i=0$ のとき $\boldsymbol{p}_i=\boldsymbol{q}$ である．このとき，i 番目の繰り返し計算における ObjectRank スコアの下限値 $\underline{\boldsymbol{r}}_t^{(i)}$ と上限値 $\overline{\boldsymbol{r}}_t^{(i)}$ を以下に定義する．

定義 3.1 (下限値)．i 番目の繰り返し計算におけるノード v の下限値は次のように計算する．

$$\underline{r}_t^{(i)}(v) = (1-d)\sum_{j=0}^{i} d^j p_t^{(j)}(v) \tag{3}$$

定義 3.2 (上限値)．i 番目の繰り返し計算におけるノード v の上限値は次のように計算する．

$$\begin{aligned}\overline{r}_t^{(i)}(v) =& (1-d)\sum_{j=0}^{i} d^j p_t^{(j)}(v)\\ &+ d^{i+1}p_t^{(i)}(v) + \frac{d^{i+1}}{1-d}\Delta_t^{(i)}\overline{A}(v)\end{aligned} \tag{4}$$

また，$\Delta_t^{(i)}$ は次のように計算する．

$$\Delta_t^{(i)}=\begin{cases}1 & (i=1)\\ \sum_{u\in V_0}\Delta_t^{(i)}(u) & (otherwise)\end{cases}$$

ただし，G_i を i 番目の繰り返し計算における部分グラフ，G_0 を元のグラフとし，V_0 は G_0 のノード集合を表す．$\Delta_t^{(i)}(v)$

は $\Delta_t^{(i)}(v) = max\{p_t^{(i)}(v) - p_t^{(i-1)}(v), 0\}$ により計算する．$\overline{\boldsymbol{A}}$ はエッジの最大の重みを格納した N 次元ベクトルであり，$\overline{A}(v) = max\{A(v,u) : u \in G_i\}$ となる．

補題 3.1 (下限値の性質)．i 番目の繰り返し計算において，下限値 $\underline{r}_t^{(i)}(v)$ は次の性質を満たす．

$$\underline{r}_t^{(i)}(v) \leqq r_t(v) \tag{5}$$

証明． 式 (2) より，

$$\begin{aligned}\boldsymbol{r}_t^{(i)} &= d\boldsymbol{A}\boldsymbol{r}_t^{(i-1)} + (1-d)\boldsymbol{q}_t \\ &= d^2\boldsymbol{A}^2\boldsymbol{r}_t^{(i-2)} + (1-d)(d\boldsymbol{A}\boldsymbol{q}_t + \boldsymbol{q}_t) \\ &= d^i\boldsymbol{A}^i\boldsymbol{r}_t^{(0)} + (1-d)\{d^{i-1}\boldsymbol{A}^{i-1}\boldsymbol{q}_t + \cdots + \boldsymbol{q}_t\} \\ &= d^i\boldsymbol{A}^i\boldsymbol{r}_t^{(0)} + (1-d)\sum_{j=0}^{i-1} d^j\boldsymbol{p}_t^{(j)}\end{aligned}$$

最終的な ObjectRank のスコアベクトルは $\boldsymbol{r}_t^{(i)}$ の収束値であるため，$\boldsymbol{r}_t = \boldsymbol{r}_t^{(\infty)}$ となる．$0 < d < 1$ かつ $\boldsymbol{A}^\infty$ の各要素は 0 以上 1 以下の値をとるため，

$$\boldsymbol{r}_t = d^\infty\boldsymbol{A}^\infty\boldsymbol{r}_t^{(0)} + (1-d)\sum_{j=0}^{\infty} d^j\boldsymbol{p}_t^{(j)} = (1-d)\sum_{j=0}^{\infty} d^j\boldsymbol{p}_t^{(j)} \tag{6}$$

が成り立つ．したがって，

$$\boldsymbol{r}_t = (1-d)\sum_{j=0}^{\infty} d^j\boldsymbol{p}_t^{(j)} \geqq (1-d)\sum_{j=0}^{i} d^j\boldsymbol{p}_t^{(j)} \tag{7}$$

が成り立つため，補題 3.1 の式 (5) が成立する． □

補題 3.2 (上限値の性質)．i 番目の繰り返し計算において，上限値 $\overline{r}_t^{(i)}(v)$ は次の性質を満たす．

$$\overline{r}_t^{(i)}(v) \geqq r_t(v) \tag{8}$$

証明． スペースの都合上により，本稿では省略する． □

また，上限値と下限値は繰り返し計算を進めると最終的に重要度の収束値に一致することが保証される．

補題 3.3 (上限値と下限値の収束性)．上限値 $\overline{r}_t^{(i)}$ と下限値 $\underline{r}_t^{(i)}$ は最終的に従来の *ObjectRank* によって得られる重要度の収束値 $r_t(v)$ に一致する．すなわち，$\underline{r}_t^{(\infty)}(v) = r_t(v) = \overline{r}_t^{(\infty)}(v)$ を満たす．

証明． 式 (3.1) より $\underline{r}_t^{(\infty)}(v) = (1-d)\sum_{j=0}^{\infty} d^j p_t^{(j)}(v)$ となるため，明らかに $\underline{r}_t^{(\infty)}(v) = r_t(v)$ が成り立つ．式 (3.2) より $\overline{r}_t^{(i)}(v) = (1-d)\sum_{j=0}^{\infty} d^j p_t^{(j)}(v) + d^\infty p_t^{(\infty)}(v) + \frac{d^\infty}{1-d}\Delta_t^{(\infty)}\overline{A}(v)$ となる．ここで，$0 < d < 1$ かつ $0 \leqq p_t^{(\infty)}(v) \leqq 1$ であるため，$d^\infty p_t^{(\infty)}(v) = 0$ となる．同様に，$0 \leqq \Delta_t^{(\infty)} \leqq 1$ かつ $0 \leqq \overline{A}(v) \leqq 1$ であるため，$\frac{d^\infty}{1-d}\Delta_t^{(\infty)}\overline{A}(v) = 0$ とある．ゆえに，$\overline{r}_t^{(\infty)}(v) = (1-d)\sum_{j=0}^{\infty} d^j p_t^{(j)}(v)$ となるため，$r_t(v) = \overline{r}_t^{(\infty)}(v)$ が成り立つ．したがって，$\underline{r}_t^{(\infty)}(v) = r_t(v) = \overline{r}_t^{(\infty)}(v)$ が成り立つ． □

Algorithm 1 Our basic proposed algorithm

Input: G, given graph; t, keyword; k, # of answer vertices
Output: Top-k vertices from final subgraph G_{i+1}

```
1: i ← 0
2: G_i ← G
3: repeat
4:     for each v ∈ G_i do
5:         calculate r_t^(i)(v) and r̄_t^(i)(v) by equation (9) and (10)
6:     end for
7:     ε_i ← the k-th largest lower bound in G_i
8:     construct G_{i+1} by pruning vertices s.t {v ∈ V_i : r̄_t^(i)(v) < ε_i} from G_i
9:     i ← i + 1
10: until |V_{i+1}| = k
11: return V_{i+1}
```

重要度評価の際の繰り返し計算において，下限値 $\underline{\boldsymbol{r}}_t^{(i)}$ と上限値 $\overline{\boldsymbol{r}}_t^{(i)}$ は次のように逐次的に計算することができる．

補題 3.4 (下限値と上限値の逐次的な計算)．i 番目の繰り返し計算において，下限値 $\underline{r}_t^{(i)}(v)$ と上限値 $\overline{r}_t^{(i)}(v)$ は次式で計算する．また，次式は $p_t^{(i)}(v)$ が計算済みであるとき $O(1)$ で計算可能である．

$$\underline{r}_t^{(i)}(v) = \begin{cases}(1-d)q_t(v) & (i=0) \\ \underline{r}_t^{(i-1)} + (1-d)d^i p_t^{(i)}(v) & (i \neq 0)\end{cases} \tag{9}$$

$$\overline{r}_t^{(i)}(v) = \begin{cases}q_t(v) + \frac{d}{1-d}\overline{A}(v) & (i=0) \\ \underline{r}_t^{(i-1)} + d^i p_t^{(i)}(v) + \frac{d^{i+1}}{1-d}\Delta_t^{(i)}\overline{A}(v) & (i \neq 0)\end{cases} \tag{10}$$

証明． $i = 0$ のとき，定義 3.1 より $\underline{r}_t^{(i)}(v) = (1-d)\sum_{j=0}^{i} d^j p_t^{(j)}(v) = (1-d)p_t^{(0)}(v) = (1-d)q_t(v)$ が成り立つ．また，定義 3.2 より $\overline{r}_t^{(i)}(v) = (1-d)q_t(v) + dq_t(v) + \frac{d}{(1-d)}\overline{A}(v) = q_t(v) + \frac{d}{(1-d)}\overline{A}(v)$ が成り立つ．このとき，d, $q_t(v)$, $\overline{A}(v)$ は定数であるため，式 (9)，(10) は $O(1)$ で計算可能である．

次に，$i \neq 0$ のとき，定義 3.1 より $\underline{r}_t^{(i)}(v) - \underline{r}_t^{(i-1)}(v) = (1-d)d^i p_t^{(i)}(v)$ となり，$\underline{r}_t^{(i)}(v) = \underline{r}_t^{(i-1)} + (1-d)d^i p_t^{(i)}(v)$ が成り立つ．また，定義 3.1，3.2 より $\overline{r}_t^{(i)}(v) - \underline{r}_t^{(i-1)}(v) = d^i p_t^{(i)}(v) + \frac{d^{i+1}}{(1-d)}\Delta_t^{(i)}\overline{A}(v)$ となり，$\overline{r}_t^{(i)}(v) = \underline{r}_t^{(i-1)}(v) + d^i p_t^{(i)}(v) + \frac{d^{i+1}}{(1-d)}\Delta_t^{(i)}\overline{A}(v)$ が成り立つ．$\underline{r}_t^{(i-1)}(v)$ は $i-1$ 番目の繰り返し計算においてあらかじめ計算されており，d, $p_t^{(i)}(v)$, $\overline{A}(v)$ は定数であるため，式 (9)，(10) は $O(1)$ で計算できる． □

3.3 基本的なアルゴリズム

提案手法の基本的なアルゴリズムを Algorithm 1 に示す．i $(i = 0, 1, \ldots)$ 番目の繰り返し計算において，各ノードの重要度の上限値 $\overline{r}_t^{(i)}(v)$，下限値 $\underline{r}_t^{(i)}(v)$ を計算する（5 行目）．閾値 ϵ_i を i 番目の繰り返し計算における k 番目に大きい下限値とする（7 行目）．このとき，上限値が ϵ_i を下回ったノードをグラフ G_i から取り除き，新たな部分グラフ G_{i+1} を構築する（8 行目）．G_{i+1} に含まれるノード数 $|V_{i+1}|$ が k に達したとき計算を終了し，k 件のノードを返す．

Algorithm 1 が従来の ObjectRank と同様の結果が得られることを示すために，以下の補題を導入する．

補題 3.5 (アルゴリズムの正確性)．提案手法は，従来の *ObjectRank* によって得られる上位 k 件のノードを正確に検索することができる．

証明． 従来の ObjectRank によって計算される重要度の収束値の k 番目に大きな値を ϵ とする．すなわち，あるノード v が上位 k 件の検索結果に含まれるとき，重要度の収束値 $r_t(v)$ は必ず $\epsilon \leqq r_t(v)$ を満たす．i 番目の繰り返し計算における k 番目に大きな下限値を ϵ_i とすると，補題 3.1 より $\epsilon_i \leqq \epsilon$ が成り立つ．提案手法は $\overline{r}_t^{(i)}(u) < \epsilon_i$ を満たすようなノード u をグラフから枝刈りするが，補題 3.2 よりノード u について $r_t(u) \leqq \overline{r}_t^{(i)}(u) < \epsilon$ が成り立つ．これは検索結果に含まれる場合の条件 $\epsilon \leqq r_t(v)$ に矛盾するため，提案手法は上位 k 件になり得ないノードを必ず枝刈りする． □

3.4 上限値の先読み

上述のアルゴリズムに加えて，我々は上限値を先読みすることで繰り返し計算の早い段階からノードの枝刈りを行い，Top-k 検索のさらなる高速化を図る．

提案手法が枝刈りするノードの数と上限値，下限値との間の関係を調べるために，DBLP の文献データベース（詳細は 4. 節で述べる）に対して予備実験を行った．Algorithm 1 に示した基本的な提案手法を実行し，各繰り返し計算におけるあるノードの上限値と下限値を図 4 に，部分グラフに含まれるノード数の推移の様子を図 5 にそれぞれ示す． 図 4 より，上限値

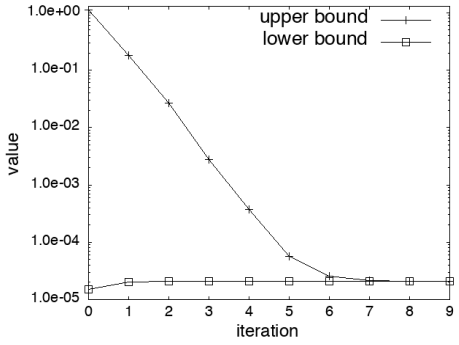

図 4 上限値と下限値の推移

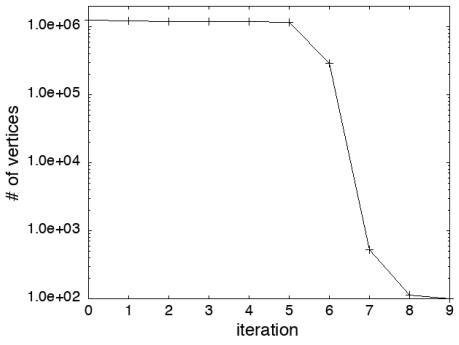

図 5 ノード数の推移

は重要度の収束値と比較して非常に大きな初期値をとり，繰り返し計算が進むにつれて指数関数的に減少していく傾向が見られた．一方で下限値は重要度の収束値と非常に近い初期値をとり，繰り返し計算の早い段階で値が収束することがわかる．また，図 5 より提案手法は繰り返し計算が進むにつれてノード数を大幅に削減できているが，1 ～ 5 番目の繰り返し計算時には約 1 % のノードに対してのみ枝刈りが行われていることがわかる．これは，Algorithm 1 に示した提案手法の枝刈り判定の条件（8 行目）に起因している．提案手法は i 番目の繰り返し計算において k 番目に大きな下限値を ϵ_i としたときに $\overline{r}_t^{(i)}(v) < \epsilon_i$ を満たすノード v の枝刈りを行うが，図 4 の $i = 0, 1, \ldots, 5$ のときのように上限値が著しく大きい場合は条件を満たす可能性が非常に低くなってしまうためである．

そこで我々は，上限値を先読み推定することで繰り返し計算の早い段階からノードの枝刈りを行い，Top-k 検索のさらなる高速化を図る．具体的には，i 番目の繰り返し計算においてユーザが指定した任意の数 s について $\overline{r}_t^{(i+s)}(v)$ を推定する．また，本稿では $\overline{r}_t^{(i+s)}(v)$ を上限値の予測値と呼ぶ．

任意の閾値 θ ($\geqq 0$) について，i 番目の繰り返し計算において $p_t^{(i)}(u) > \theta$ となるノードの集合を $\mathbb{P}_t^{(i)}$，$\mathbb{P}_t^{(i)}$ から s ホップ以内に到達不可能であるノードの集合を $\overline{\mathbb{R}}_t^{(i+s)}$ とする．このとき，i 番目の繰り返し計算において $v \in \overline{\mathbb{R}}_t^{(i+s)}$ となるノード v の上限値の予測値は以下の式で計算する．

定義 3.3 (上限値の予測値)．i 番目の繰り返し計算において，任意の自然数 s (> 0) について $v \in \overline{\mathbb{R}}_t^{(i+s)}$ を満たすノード v の上限値の予測値 $\overline{r}_t^{(i+s)}(v)$ は以下の式で計算される．

$$\overline{r}_t^{(i+s)}(v) = \underline{r}_t^{(i)} + \frac{d^{i+s+1}}{1-d}\Delta_t^{(i)}\overline{A}(v) \tag{11}$$

式 (11) によって得られる上限値の予測値は以下の性質を満たす．

補題 3.6 (上限値の予測値の性質)．i 番目の繰り返し計算において $\overline{r}_t^{(i)}(v)$ と $\overline{r}_t^{(i+s)}(v)$ は次の性質を満たす．

$\theta = 0$ であるとき，

$$r_t(v) \leqq \overline{r}_t^{(i+s)}(v) \leqq \overline{r}_t^{(i)}(v) \tag{12}$$

$\theta > 0$ であるとき，

$$\overline{r}_t^{(i+s)}(v) \leqq \overline{r}_t^{(i)}(v) \tag{13}$$

証明． 式 (10) より，$\overline{r}_t^{(i+s)}(v) = \underline{r}_t^{(i)} + d^{i+s}p_t^{(i+s)}(v) + (1-d)d^{i+1}\{\sum_{k=1}^{s-1} d^{k-1}p_t^{(i+k)}(v)\} + \frac{d^{i+s+1}}{1-d}\Delta_t^{(i+s)}\overline{A}(v)$ が得られる．$\boldsymbol{p}_t^{(i+s)} = \boldsymbol{A}^{i+s}\boldsymbol{q}_t$ より $\boldsymbol{p}_t^{(i+s)} = \boldsymbol{A}\boldsymbol{A}^{i+s-1}\boldsymbol{q}_t = \boldsymbol{A}^s\boldsymbol{p}_t^{(i)}$ となるため，

$\theta = 0$ であるとき，$\overline{\mathbb{R}}_t^{(i+s)}$ に含まれるノード v について $p_t^{(i+1)}(v), p_t^{(i+2)}(v), \ldots, p_t^{(i+s)}(v)$ はすべて 0 となる．ゆえに，$\overline{r}_t^{(i+s)}(v) = \underline{r}_t^{(i)} + \frac{d^{i+s+1}}{1-d}\Delta_t^{(i+s)}\overline{A}(v)$ となる．したがって，補題 3.2 と $\Delta_t^{(i+s)} \geqq \Delta_t^{(i)}$ より，$r_t^{(i)} \leqq \overline{r}_t^{(i+s)}(v) \leqq \overline{r}_t^{(i)}(v)$ が成り立つ．

$\theta > 0$ であるとき，$\overline{\mathbb{R}}_t^{(i+s)}$ に含まれるノード v について $p_t^{(i+1)}(v), p_t^{(i+2)}(v), \ldots, p_t^{(i+s)}(v)$ は非零の値を取りうる．ゆえに，式 (11) は従来の ObjectRank により得られる重要度の収束値 $r_t(v)$ を下回る可能性があり，上限値としての性能を満たさなくなる． □

3.5 先読みを用いたアルゴリズム

上限値の先読みを用いた提案手法のアルゴリズムを Algorithm 2 に示す．基本的な流れは Algorithm 1 と同様であるが，7 ～ 12 行目で上限値を先読みする点が異なる．また，入力として先読みを行う回数 s，閾値 θ，s ホップでの到達可能性を格納した索引 $\boldsymbol{R}$ を与える．$\boldsymbol{R}$ は，グラフに含まれる各ノードについて，s ホップ以内で到達可能なノード集合を計算し格納したものである．

i ($i = 0, 1, \dots$) 番目の繰り返し計算において，各ノードの重要度の上限値 $\overline{r}_t^{(i)}(v)$，下限値 $\underline{r}_t^{(i)}(v)$ を計算する（4 ～ 6 行目）．次に，$p_t^{(i)} > \theta$ を満たすノード集合 $\mathbb{P}_t^{(i)}$，$\mathbb{P}_t^{(i)}$ から s ホップ以内で到達不可能であるノード集合 $\overline{R}_t^{(i+s)}$ を計算する（7, 8 行目）．$\overline{\mathbb{R}}_t^{(i+s)}$ に含まれる各ノード v について，$\overline{r}_t^{(i+s)}(v)$ を計算し枝刈り判定に用いる上限値に置き換える（9 ～ 12 行目）．閾値 ϵ_i を i 番目の繰り返し計算における k 番目に大きい下限値とする（13 行目）．このとき，上限値が ϵ_i を下回ったノードをグラフ G_i から取り除き，新たな部分グラフ G_{i+1} を構築する（14 行目）．G_{i+1} に含まれるノード数 $|V_{i+1}|$ が k に達したとき計算を終了し，k 件のノードを返す．

上限値の先読みを用いた場合の提案手法（Algorithm 2）の正確性について，以下の補題が成り立つ．

補題 3.7 (アルゴリズムの正確性)．*Algorithm 2* は従来の *ObjectRank* によって得られる上位 k 件のノードと比較して，$\theta > 0$ であるときは近似解を，$\theta = 0$ であるときは同一の解を導出する．

証明. 従来の ObjectRank によって計算される重要度の収束値の k 番目に大きな値を ϵ とする．すなわち，あるノード v が上位 k 件の検索結果に含まれるとき，重要度の収束値 $r_t(v)$ は必ず $\epsilon \leqq r_t(v)$ を満たす．i 番目の繰り返し計算における k 番目に大きな下限値を ϵ_i とすると，補題 3.1 より $\epsilon_i \leqq \epsilon$ が成り立つ．提案手法は $\overline{r}_t^{(i)}(u) < \epsilon_i$ を満たすようなノード u をグラフから枝刈りするが，補題 3.2, 3.6 より，$\theta = 0$ であるときはノード u について $r_t(u) \leqq \overline{r}_t^{(i+s)}(u) \leqq \overline{r}_t^{(i)}(u) < \epsilon$ が成り立つ．これは検索結果に含まれる場合の条件 $\epsilon \leqq r_t(v)$ に矛盾するため，提案手法は上位 k 件になり得ないノードを必ず枝刈りする．$\theta > 0$ であるとき，$\overline{r}_t^{(i+s)}(u) < \epsilon \leqq r_t(v)$ を満たす可能性があるため，上位 k 件に含まれるノードを枝刈りしてしまう場合がある． □

4. 評 価 実 験

本節では，提案手法，従来の ObjectRank および既存手法の BinRank [13] を実行し，効率性と検索結果上位 k 件の近似精度の観点で提案手法の有効性を検証する．

- **提案手法:** Algorithm 1 及び Algorithm 2 に示した提案手法．
- **ObjectRank:** べき乗法を用いた従来の計算方法 [2].
- **BinRank:** 全体のグラフと比較してノード数の少ない部分グラフを構築することで高速化を図る手法 [13]．BinRank は，データベースに含まれるキーワードをクラスタリングし，クラスタに含まれるキーワードとの関連度が著しく低いノードを取り除いた部分グラフを，クラスタごとに構築し索引に格納する．クエリ処理時には，クエリキーワードを含むクラスタから作成された部分グラフのみを対象に，べき乗法による重要度評価を行う．

Algorithm 2 Our advanced proposed algorithm

Input: G, given graph; t, keyword; k, # of answer vertices; s, the constant; θ, threshold; $\boldsymbol{R}$, reachability index
Output: Top-k vertices from final subgraph G_{i+1}

1: $i \leftarrow 0$
2: $G_i \leftarrow G$
3: **repeat**
4: **for each** $v \in G_i$ **do**
5: calculate $\underline{r}_t^{(i)}(v)$ and $\overline{r}_t^{(i)}(v)$ by equation (9) and (10)
6: **end for**
7: add vertices $s.t.$ $\{v \in V_i : p_t^{(i)}(v) > \theta\}$ to $\mathbb{P}_t^{(i)}$
8: compute $\overline{\mathbb{R}}_t^{(i+s)}$ from $\mathbb{P}_t^{(i)}$ and reachability table $\boldsymbol{R}$
9: **for each** $v \in \overline{\mathbb{R}}_t^{(i+s)}$ **do**
10: calculate $\overline{r}_t^{(i+s)}(v)$ by Equaeion (11)
11: $\overline{r}_t^{(i)}(v) \leftarrow \overline{r}_t^{(i+s)}(v)$
12: **end for**
13: $\epsilon_i \leftarrow$ the k-th largest lower bound in G_i
14: construct G_{i+1} by pruning vertices $s.t.$ $\{v \in V_i : \overline{r}_t^{(i)}(v) < \epsilon_i\}$ from G_i
15: $i \leftarrow i + 1$
16: **until** $|V_{i+1}| = k$
17: **return** V_{i+1}

表 2 DBLP データセットの詳細

総ノード数	1,238,266
総エッジ数	5,149,294
Conference ラベルのノード数	12,609
Year ラベルのノード数	67
Paper ラベルのノード数	629,814
Author ラベルのノード数	595,776
Conference, Year ノード間のエッジ数	48
Year, Paper ノード間のエッジ数	1,259,628
Paper, Paper ノード間のエッジ数	1,265,502
Paper, Author ノード間のエッジ数	2,624,116

データセットは AMiner (注1) が公開している文献データベースを用いた．Schema Graph は図 2 に示した文献 [2] と同様のものを使用し，Data Graph を構築した．Data Graph の詳細は表 2 に示す．ダンピングファクタ d は文献 [2] と同様に $d = 0.85$ とした．プログラムは C++ で実装し，計算機は 3.5GHz CPU，メモリが 128GB の Linux サーバを用いた．

4.1 効 率 性

上述の 3 つの手法をそれぞれ実行し，実行時間を評価した．提案手法は，Algorithm 1 に示した基本的なアルゴリズム（Alg. 1）と Algorithm 2 に示した先読みを用いたアルゴリズム（Alg. 2）の両方を実行し，Alg. 2 については s と θ の値を変化させて実行時間の計測を行った．結果を図 6 に示す．実験結果より，逐次枝刈りによる提案手法（Alg. 1）は従来のべき乗法を用いた ObjectRank と比較して，約 5 倍程度高速であることが示された．ObjectRank はすべてのノードの重要度が収束するまで繰り返し計算を行う必要があるが，提案手法は解になり得ないノードを逐次的に枝刈りし上位 k 件のノードが得られた時点で計算を終了するため，高速な計算が可能となる．提案手法（Alg. 1）既存手法である BinRank と比較してもより高速

(注1)：http://aminer.org

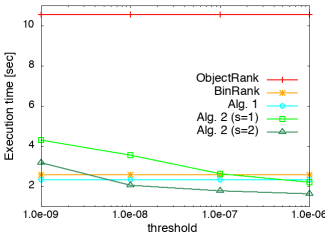

図 6　実 行 時 間

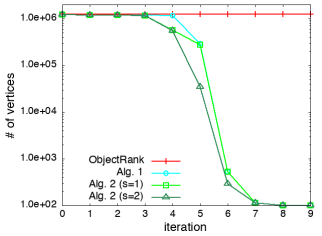

図 7　ノード数の遷移 ($\theta = 1.0e-8$)

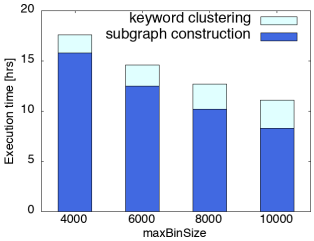

図 8　BinRank の事前計算

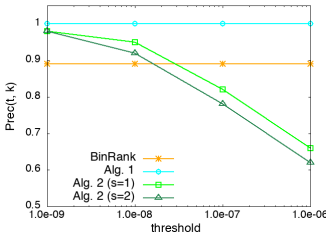

図 9　近 似 精 度

に結果を導出できることが示された．先読みを行った場合の提案手法（Alg. 2）は，θ の値を大きくするにつれて，実行時間が減少することがわかった．これは，θ が大きくなるにつれて $\mathbb{P}_t^{(i)}$ に含まれるノード数が減少することに起因する．先読みを行う対象のノードは $\mathbb{P}_t^{(i)}$ に含まれるノードから s ホップ以内に到達不可能なノード集合 $\overline{\mathbb{R}}_t^{(i+s)}$ であるため，$\mathbb{P}_t^{(i)}$ のノード数が減少するにつれて $\overline{\mathbb{R}}_t^{(i+s)}$ のノード数が増加し，より多くのノードに対して上限値の先読みを行うことができるようになる．また，$s = 1$ と比較して $s = 2$ の方が高速であることがわかった．これは，s の値が大きいほどより先の繰り返し計算時における上限値を先読みできるため，枝刈りの条件に合致する場合が増加するためである．

図 7 は，提案手法と ObjectRank の各イテレーションにおけるノード数の遷移の様子を示している．実験結果より，ObjectRank は各イテレーションですべてのノードの重要度を計算する一方で，提案手法はイテレーション数が増えるにつれてノード数を大幅に削減できていることわかる．先読みを行った場合（Alg. 2），先読みを行わない場合（Alg. 1）と比較して，$i = 36$ 番目の繰り返し計算時においてより多くのノードを枝刈りすることが可能となる．

図 8 は BinRank の事前計算部の実行時間を示している．$maxBinSize$ はクラスタサイズを制御する内部パラメータであり，$maxBinSize$ が大きくなるほどクラスタの数が減少し，索引に格納される部分グラフの数も減少する．実験結果より，BinRank は 10 時間以上もの事前計算を要することが示された．

表 3　$\boldsymbol{R}$ の計算時間

s	Execution time（sec）
1	0.8
2	353442
3	dnf

上限値の先読みを行う場合，グラフに含まれる各ノードについて，s ホップ以内で到達可能なノード集合を事前計算し，索引 $\boldsymbol{R}$ として保持する必要があった．表 3 に $\boldsymbol{R}$ の計算にかかる実行時間を示す．$s = 1$ の場合は遷移行列を参照するだけで $\boldsymbol{R}$ を構築できるため，高速に $\boldsymbol{R}$ を計算することができる．$s = 2$ の場合は各頂点を始点として幅優先探索を実行し s ホップ以内で到達できるノードを探索するため，索引の構築には約 6 分の時間を要する結果となった．s を大きくするにつれて $\boldsymbol{R}$ が密になってしまうため，$s = 3$ 以上の場合はメモリ不足により実行不可能となってしまった．BinRank の索引構築と比較しても事前計算にかかる時間は著しく小さいことが示されたが，$\boldsymbol{R}$ の効率的な計算は今後の重要な課題として挙げられる．

4.2　近 似 精 度

ObjectRank によって得られた結果を真値とし，提案手法と BinRank の近似精度を計測した．近似精度の指標には式 (14) を用いた．

$$Prec(t,k) = \frac{|CompSet(t,k) \cap ORSet(t,k)|}{k} \quad (14)$$

$CompSet(t,k)$ はキーワード t に対して提案手法または BinRank を，$ORSet(t,k)$ は ObjectRank をそれぞれ実行した場合に得られた上位 k のノードの集合を表す．

結果を図 9 に示す．実験結果より，提案手法は上限値の先読みを行わない場合（Alg. 1），ObjectRank と全く同一の結果が得られることが示された．補題 3.5 より明らかであるが，提案手法は重要度の上限値と下限値を推定することで上位 k 件になり得ないノードを発見することができるためである．一方，上限値の先読みを行う場合（Alg. 2）は得られる結果は近似解となり，θ の値を大きくするにつれて近似精度が低下することがわかった．これは，θ の値が大きいほど式 (11) が $i+s$ 番目の繰り返し計算時に本来計算される値から異なる値をとるようになってしまうためである．同様の理由から，s の値を大きくするほど近似精度は低下してしまうことがわかる．

効率性と近似精度の実験より，上限値の先読みを用いた提案手法は s と θ の値を大きくするほど実行時間が減少し，近似精度が低下することがわかった．実行時間と近似精度の間にはトレードオフの関係が存在するため，適切なパラメータの選択方法は今後の課題として挙げられる．

5.　関 連 研 究

PageRank や ObjectRank といったランダムウォークモデルに基づくグラフ分析技術の研究が近年盛んに行われている．既存手法は (1) ノードの枝刈りに基づく手法と (2) 索引を用いた手法の 2 つに分けることができる．

ノードの枝刈りに基づく手法: これらの手法は，グラフから不必要なノードや他のノードの重要度に与える影響が少ないノードを推定し，枝刈りすることで高速化を図る．Sakakura [14] らは，グラフ全体ではなく局所的なグラフを構築することで重要度を推定する手法を提案した．ObjectRank はエッジごとに異なる重みをもち，重みが小さいエッジをもつノードは他のノードへと与える影響も小さい．坂倉らの手法は，エッジの重みを

考慮することで重要度推定に与える影響の小さいノードを推定し，グラフから枝刈りすることで高速な重要度推定を実現した．Fujiwara [7] らは，繰り返し計算の過程で解になり得ないノードを逐次的に枝刈りし，PageRank の Top-k 検索を高速化する手法 F-Rank を提案した．F-Rank では，繰り返し計算の各ステップにおいて各ノードの重要度の上限値と下限値の推定値を用いて解になり得ないノードを特定し，グラフから枝刈りする．グラフからノードの枝刈りを行うが，検索結果は理論的に正しいことが保障されている．しかし，F-Rank は遷移行列の列が正規化される PageRank の特性を利用しているため，ObjectRank には適用できない．

索引を用いた手法: クエリ応答の効率化のために，Chakrabarti ら，Hwang らは索引を用いて高速化を図る手法 HubRank [4]，BinRank [13] をそれぞれ提案した．これらの手法は索引を事前に構築し，クエリ処理時には索引を参照することで検索結果を近似することでクエリ処理時間を短縮する．HubRank は，*hub* ノードと呼ばれるクエリログに基づいて選択された一部のノードの *random walk fingerprints* [6] を事前計算し索引付けする．クエリ処理時には，クエリキーワードを含むノードから幅優先探索を開始し，*hub* ノードに到達するかクエリノードから一定の距離が離れた時点で探索をやめ，得られた部分グラフについて索引に基づいた重要度推定を行う．BinRank は，データベースに含まれるキーワードをクラスタリングし，クラスタに含まれるキーワードとの関連度が著しく低いノードを取り除いた部分グラフを，クラスタごとに構築し索引に格納する．クエリ処理時には，クエリキーワードを含むクラスタから作成された部分グラフのみを対象に，べき乗法による重要度評価を行う．

しかし，これらの手法は索引構築に膨大な時間を要するだけでなく，クエリ応答時にも重要度推定のための計算を行う必要があるという問題がある．我々が行った評価実験では，BinRank は索引構築に 10 時間以上かかることが示された（図 8）．

既存手法とは異なり，我々の提案手法は大規模なグラフを対象とした場合でも高速な Top-k 検索が可能である．我々が行った評価実験により，重要度先読みを行わない場合は従来の ObjectRank と同様の結果を得ることができることに加え，BinRank と同程度の時間で結果を得られることが示された．

6. 結　　論

本稿では ObjectRank の Top-k 検索の高速化手法を提案した．提案手法では，繰り返し計算の過程で各ノードの重要度の取り得る上限値と下限値を推定することで，上位 k 件になり得ないノードをグラフから逐次的に枝刈りする．この手法により重要度評価の繰り返し計算を進めるにつれてノードが減少するため，従来の ObjectRank と比較して Top-k 検索を高速に行うことができる．また，提案手法は一部のノードの上限値を任意の回数だけ先読みすることで，より高速な枝刈りを実現する．実データを用いた評価実験では，提案手法は従来のべき乗法を用いた ObectRank および既存手法である BinRank と比較して高速に検索結果が得られることを示した．

今後の課題として，アルゴリズムのチューニングが挙げられる．現在の実装では，s ホップでの到達可能性の計算がボトルネックとなってしまい，ノードの枝刈りを高速に行うことができる一方で全体の実行時間にあまり変化しないことがわかった．s ホップでの到達可能性の計算をより高速に行うことで，さらなる高速化が期待できる．

謝　　辞

本研究は，JSPS 科研費 16H06650 ならびに JST ACT-I の助成を受けたものである．

文　　献

[1] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast Incremental and Personalized PageRank. *Proc. VLDB*, Vol. 4, No. 3, pp. 173–184, 2010.

[2] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: Authority-Based Keyword Search in Databases. In *Proc. VLDB*, pp. 564–575, 2004.

[3] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proc. WWW*, pp. 107–117, 1998.

[4] Soumen Chakrabarti. Dynamic Personalized Pagerank in Entity-Relation Graphs. In *Proc. WWW*, pp. 571–580, 2007.

[5] Paul-Alexandru Chirita, Stefania Costache, Wolfgang Nejdl, and Raluca Paiu. Beagle＋＋: Semantically Enhanced Searching and Ranking on the Desktop. In *Proc. ESWC*, pp. 348–362, 2006.

[6] Dniel Fogaras, Balzs Rcz, Kroly Csalogny, and Tams Sarls. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics*, Vol. 2, No. 3, pp. 333–358, 2005.

[7] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Fast and Exact Top-k Algorithm for PageRank. In *Proc. AAAI*, pp. 1106–1112, 2013.

[8] Stefania Ghita, Wolfgang Nejdl, and Raluca Paiu. Semantically Rich Recommendations in Social Networks for Sharing, Exchanging and Ranking Semantic Context. In *Proc. ISWC*, pp. 293–307, 2005.

[9] Marco Gori and Augusto Pucci. ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines. In *Proc. IJCAI*, pp. 2766–2771, 2007.

[10] Vagelis Hristidis, Heasoo Hwang, and Yannis Papakonstantinou. Authority-Based Keyword Search in Databases. *ACM Trans. Database Syst.*, Vol. 33, No. 1, 2008.

[11] Vagelis Hristidis, Yao Wu, and Louoiqa Raschid. Efficient Ranking on Entity Graphs with Personalized Relationships. *TKDE*, Vol. 26, No. 4, pp. 850–863, April 2014.

[12] Heasoo Hwang, Andrey Balmin, Hamid Pirahesh, and Berthold Reinwald. Information Discovery in Loosely Integrated Data. In *Proc. SIGMOD*, pp. 1147–1149, 2007.

[13] Heasoo Hwang, Andrey Balmin, Berthold Reinwald, and Erik Nijkamp. BinRank: Scaling Dynamic Authority-Based Search Using Materialized SubGraphs. In *Proc. ICDE*, pp. 66–77, 2009.

[14] Yuta Sakakura, Yuto Yamaguchi, Toshiyuki Amagasa, and Hiroyuki Kitagawa. A Local Method for ObjectRank Estimation. In *Proc. IIWAS*, p. 92. ACM, 2013.

[15] Satu Elisa Schaeffer. Graph Clustering. *Computer Science Review*, Vol. 1, No. 1, pp. 27–64, 2007.

[16] Yuto Yamaguchi, Tsubasa Takahashi, Toshiyuki Amagasa, and Hiroyuki Kitagawa. TURank: Twitter User Ranking Based on User-Tweet Graph Analysis. In *Proc. WISE*, pp. 240–253, 2010.