1K-06

大規模グラフに対する逐次的なノードの枝刈りを用いた ObjectRank の高速化

佐藤 朋紀† 塩川 浩昭‡ 北川 博之‡

† 筑波大学情報学群情報科学類 ‡ 筑波大学計算科学研究センター

1 はじめに

ObjectRank [1] は，データベース内の各オブジェクトの重要度を評価し，キーワード検索を実現する分析手法である．代表的なグラフ分析手法の一つである PageRank を拡張した手法であり，複数種類のノードやエッジからなるグラフを扱うことができる．しかし，ObjectRank は行列ベクトル積を繰り返し計算し，グラフ上の全てのノードの重要度を評価する必要がある．したがって，大規模グラフを対象とした際の計算コストが問題となる．

そこで本稿では，ObjectRank を高速化するアルゴリズムを提案する．提案手法では，重要度が著しく低いノードを逐次的に計算対象から枝刈りすることで ObjectRank の高速化を図る．本稿では，提案手法の概要を述べるとともに，文献データベースを用いた評価実験の結果を示す．

2 ObjectRank

ObjectRank は，データベース上のオブジェクトをグラフとして表現するために，Authority Transfer Schema Graph を定義する（以降，Schema Graph と呼ぶ）．Schema Graph はグラフのノードとエッジの種類，及び各エッジの重みを定義したグラフである．図 1 に文献データベースを表現した Schema Graph の例を示す．図 1 の例では，”Conference” や “Year” といった 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．次に，Schema Graph に基づき対象のデータから Authority Transfer Data Graph を構築する（以降，Data Graph と呼ぶ）．Data Graph におけるエッジの重みは，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，出次数は同じ種類のエッジのみに限定する．図 2 は，図 1 に基づいて作成された Data Graph の例である． ノード “Balmin, A.” から “ObjectRank” には重みが 0.1 であるエッジが張られている．これは，Schema

図 1 文献データベースの Schema Graph

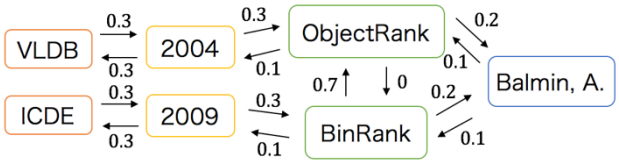

図 2 Data Graph の例

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．Data Graph の遷移行列を $\boldsymbol{A}$ とし，$\boldsymbol{A}$ の (i,j) 要素に Data Graph 内のノード v_j から v_i にエッジが存在する場合にその重みの値を，それ以外の場合は 0 を格納する．このとき，各ノードの重要度を示すベクトル $\boldsymbol{r} = [r(v_1), r(v_2), \ldots, r(v_n)]^T$ は，次式をべき乗法により，r が収束するまで反復計算を行うことで求める．

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

ただし，d はダンピングファクタ，$BS(t)$ をクエリとして与えられたキーワード t を含むノード集合としたとき，$\boldsymbol{q}$ は n 次元のクエリベクトルであり，ノード v が t を含む場合 $q(v) = 1/|BS(t)|$，それ以外の場合は $q(v) = 0$ となる．

3 関連研究

Fujiwara ら [2] は，繰り返し計算の過程で解になり得ないノードを枝刈りする PageRank の Top-k 検索高速化手法 F-Rank を提案した．F-Rank は遷移行列の列が正規化される PageRank の性質を利用するため，ObjectRank には適用できない．Hwang ら [3] は，元のグラフから比較的ノード数の少ない部分グラフを構築することで ObjectRank を高速化する BinRank を提案した．BinRank はクエリ処理時にべき乗法による反復計算で最終的な重要度を決定する必要があり，依然として計算コストが問題となる．

Fast ObjectRank for Large Graphs via Incremental Node Pruning

Tomoki Sato †, Hiroaki Shiokawa ‡ and Hiroyuki Kitagawa ‡

† College of Information Science, University of Tsukuba

‡ Center for Computational Sciences, University of Tsukuba

4 提案手法

ObjectRank は，重要度の著しく低いノードは検索結果に影響を持ちにくく，本来計算する必要がないという性質がある．そこで提案手法では，繰り返し計算の過程で重要度が著しく低いノードを特定し，逐次的に計算対象から枝刈りすることによって ObjectRank の高速化を図る．

枝刈り可能なノードを特定するために，提案手法では各ノードが取り得る重要度 $\boldsymbol{r}$ の上限値を推定する．ノードの上限値がユーザの設定した閾値を下回ったとき，そのノードは検索結果に現れないことが保証されるため，重要度の計算対象から枝刈りをする．計算を進めるにつれて，計算対象となるノードが逐次的に減少するため，ObjectRank と比較して高速に結果を計算できる．重要度の上限値は定義 1 に基づいて計算する．

[定義 1] (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 $\overline{r}_i(v)$ は次のように計算する．

$$\overline{r}_i(v) = (1-d)\Sigma_{j=0}^{i} d^j s_j(\mathrm{v}) + d^{i+1} s_i(\mathrm{v}) + (1-\mathrm{d})\frac{d^{i+1}}{(1-\alpha_{max})^2}\Delta_i \bar{A}(v) \quad (3)$$

$\boldsymbol{s}_i$ は長さ i のランダムウォークの確率を表す n 次元ベクトルであり，$\boldsymbol{s}_i = \boldsymbol{A}^i\boldsymbol{q}$ で計算する． α_{max} は 0 以上 1 以下の値を持ち，Schema Graph 内のノードの出エッジの重みの総和の最大値をとる．図 1 の例の場合，”Paper” の出エッジの重みの総和は 1.0 であり Schema Graph 内で最大であるため，$\alpha_{max} = 1.0$ となる．また，Δ_i は次のように計算する．

$$\Delta_i = \begin{cases} 1 & (i=0) \\ \Sigma_{u \in G_i}\Delta_i(u) & (i \neq 0) \end{cases} \quad (4)$$

ただし，元のグラフを G_0 としたとき， G_i は $i\,(i = 0,1,\dots)$ 番目の繰り返し計算における計算対象の部分グラフである． Δ_i は $\Delta_i(u) = \max\{s_i(u) - s_{i-1}(u), 0\}$ で計算する．$\bar{\boldsymbol{A}}$ 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， $\bar{A}(v) = \max\{A[v,u] : u \in G\}$ となる．

次に，提案手法のアルゴリズムについて述べる．Algorithm1 に提案手法の疑似コードを示す．提案手法では，定義 1 に基づいて逐次的なノードの枝刈りを行う．具体的には， $i\,(i = 0,1,\dots)$ 番目の繰り返し計算において各ノード $u\,(u \in G_i)$ の重要度 $r_i(u)$，重要度の上限値 $\overline{r}_i(u)$をそれぞれ計算する．このとき，上限値 $\overline{r}_i$ が閾値を下回ったノードをグラフ G_i から枝刈りし，新たな部分グラフ G_{i+1} を構築する．i 番目の繰り返し計算において，G_i に含まれる全てのノードの重要度が収束したとき，または，ユーザが設定した繰り返し計算の上限回数に達したとき，アルゴリズムを終了する．

Algorithm 1 提案手法

Input: G, 元のグラフ; ϵ, 閾値; max_iteration, 最大反復回数
Output: 枝刈りされたものを除いた各ノードの重要度

```
i ← 0
G_i ← G
while i < max_iteration do
    ε ← ε/G_i のノード数
    for each G_i に含まれるノード v に対して do
        重要度 r_i(v)，上限値 r̄_i(v) を計算
    end for
    if G_i の全てのノードの重要度が収束 then
        アルゴリズムを終了する
    end if
    G_i から上限値が ε を下回るノードを取り除き G_{i+1} を構築
    i ← i + 1
end while
```

5 評価実験

提案手法の実行時間を DBLP の文献データベース (https://aminer.org/billboard/citation) を用いて評価した．ノード数とエッジ数はそれぞれ 1,238,266，5,149,294 である．実験は Intel Xeon 3.5GHz , 128GB RAM の Linux サーバを用いた．また，d は 0.85 に固定した．実験結果を表 1 に示す．ϵ が 0，0.1 であるとき，ObjectRank の方が高速であった．これは，閾値が小さい場合は枝刈りが行われにくく，上限値推定にオーバーヘッドがかかってしまったためである．一方，ϵ が 0.2，0.3 であるとき，枝刈りがうまく働いたことにより提案手法の方が高速となった．

表 1. 実行時間の比較

手法	実行時間 (ms)
従来手法（ObjectRank）	10605
提案手法（$\epsilon = 0.0$）	16061
提案手法（$\epsilon = 0.1$）	12906
提案手法（$\epsilon = 0.2$）	1399
提案手法（$\epsilon = 0.3$）	1263

まとめ

本稿では，各ノードが取り得る重要度の上限値を推定し，検索結果になり得ないノードを逐次的にグラフから枝刈りすることで ObjectRank を高速化する手法を提案した．実データを用いた評価実験により，提案手法が ObjectRank より高速に計算できることが示された．

謝辞

本研究は，JSPS 科研費(26280037,16H07410)の助成を受けたものである．

参考文献

[1] Balmin, A. Hristidis, V. Papakonstantinou, Y. “ObjectRank: Authority-Based Keyword Search in Databases” In Proc. VLDB 2004.
[2] Fujiwara et al., “Fast and Exact Top-k Algorithm for PageRank” In Proc. AAAI 2013
[3] Hwang et al., “BinRank: Scaling Dynamic Authorty-Based Search Using Materialized SubGraphs” In Proc. ICDE 2009