グラフデータベースにおける正確なTop-kキーワード検索方式(O)

蒋　　吏君† 大森　　匡† 星　　守†

† 電気通信大学大学院情報システム学研究科　〒182–8585 東京都調布市調布ヶ丘1–5–1
E-mail: †{jiang,omori}@hol.is.uec.ac.jp

あらまし　著者らは, グラフデータベースにおけるtop-kキーワード検索方式の従来研究[1] [4]を冗長解抑制を伴った正確に上位k個の解を列挙できる算法に改善する. 従来研究DPBFには冗長解問題とtop-k解の最適解問題があるため, 正確なtop-1解しか出せない. MDPBF法[4]はDPBF法[1]の改良としてtop-k解 の最適解問題を解決したが, DPBF法とMDPBF法に存在する冗長解を解決できなかった. 本稿でそれらの冗長解を定義し, その抑制方法を提案する. 実装できた冗長解抑制付きMDPBFアルゴリズムの性能評価も報告する.

キーワード　グラフデータベース，top-kキーワード検索，冗長解抑制

Exact Top-k Keyword Search Algorithms on Graph Databases

Lijun JIANG†, Tadashi OHMORI†, and Mamoru HOSHI†

† The University of Electro-Communications,Graduate School of Information Systems　Chofugaoka 1–5–1, Chofu, Tokyo, 182–8585 Japan
E-mail: †{jiang,omori}@hol.is.uec.ac.jp

Abstract　We optimize top-k keyword search methods on graph databases from the past researches [1] [4] to exact top-k keyword search algorithms with redundant answer elimination. As the past research, DPBF [1] can only find out the first answer exactly in top-k answers. MDPBF [4] resolves the best answer problem in DPBF. In this paper, we define redundant answers in DPBF [1], MDPBF [4], and propose the methods to eliminate them. We report performance evaluation of MDPBF algorithm with redundant answer elimination.

Key words　Graph Database, top-k keyword search, redanduncy elimination

1. 研究背景と目的

近年, 関係データベースのような構造化データやXMLなどの半構造データを対象とするキーワード検索方式がデータベースの研究として注目されている. この研究のうち, グラフでデータベースを表して検索を行う方法がある. 関係データベースのタプルやXMLのタグをノード, 関係データベースの外部キー参照やXMLの階層関係をエッジと見なせば, 全てのデータを一つの巨大なグラフに統合して表せる. 本稿はグラフデータベースにおけるtop-kキーワード検索問題を扱う.

グラフデータベースにおけるキーワード検索問題の一般的な解決法は二つある. 一つは, l個のキーワードを入力として, 高々l個の葉ノード集合から唯一の根ノードへの最短経路を併合して解とする方法 (代表的なアルゴリズムはBANK-I/II [3] [5], BLINKS [6]) である. もう一つは, 当該葉ノードを含む連結木に対して成長操作と併合操作を行ってコスト (グラフのエッジの重み総和) 昇順で列挙する方法 (代表的なアルゴリズムはDPBF [1]) である. BANKS-I/IIでは, 各葉ノードから成長させた最短経路をルートノードだけで併合する. そのため, 上位k個の解は全て近似解となる可能性がある. DPBFでは, ノードごとに連結木の成長操作と併合操作を行う. これにより, steiner木としてのコスト最小の正確な解を計算できるが, 二番目から上位k個の解を列挙する際に近似的に計算するしかないうえ, 得られた解に対して正確に評価されなかったので品質が不明である.

昨年我々は, DPBFの戦略に基づいてtop-kの解を厳密に計算できるMDPBF [4]を提案した. コスト昇順でTop-kの解の正確さを保証できるが, DPBFの同型木の冗長列挙や, 解となる連結木と関係のない空間への無駄な探索により計算速度の著しい低下という二つの問題がまだ残っている. また, MDPBFは冗長解列挙の抑制に関して考慮しなかった.

本稿では, 冗長解列挙を回避した冗長抑制付きDPBFに作り直した後, MDPBF法の戦略に基づいて正確なtop-kの解の問題を解決した冗長抑制付きMDPBFを提案する.

以下, 2節でDPBFの概要を述べる. 3節でDPBF法の問題の定義と回避方法を述べ, DPBFの原論文のサンプルグラフで

冗長抑制付き DPBF の解を評価する．4 節で MDPBF の概要を述べる．5 節で MDPBF における冗長解の定義と抑制法を述べ, 冗長抑制付き MDPBF 法を提案する．6 節では本稿より提案される冗長抑制付き MDPBF の時間計算量と記憶量を述べる．7 節で性能評価を行う．8 節でまとめる．

2. DPBF の概要 [1]

グラフでのキーワード検索は GST-k [2](Group Steiner Tree) を求める問題である．GST-k を求める問題は本質的に最小集合被覆問題と同等であり, NP 困難である．

DPBF 法 [1] は, 任意のデータグラフから全てのキーワードを含む Top-k 個の最小コスト連結木を求めるための手法である．最小コスト連結木とはキーワードを直接含むノードが繋がっていてエッジの重み総和の最も小さい連結木である．エッジの重みが小さいほど, ノード間の関係が強いとする．DPBF 法の (最悪) 計算量 (Fibonacci Heap を使う場合) はキーワード l に対して, $O(3^l n + 2^l((l+\log n)n+m))$($l$: キーワード数, n : ノード数, m : エッジ数) となる．[1]

DPBF 法は以下の漸化式の計算を, キーワード全集合 $\mathbf{P}(=\{\mathrm{p_1}, \mathrm{p_2}, ..., \mathrm{p_l}\})$ を含む連結木が k 個見つかるまで繰り返すことによって, GST-k の解を求める．($\mathbf{p}, \mathbf{p_1}, \mathbf{p_2}, ...$ は $\mathbf{P}$ の, 空でない任意の部分集合とする)

(ノード v がキーワード部分集合 $\mathbf{p}$ を直接含むとき)

$$T_o(v, \mathbf{p}) = v \tag{1}$$

(それ以外の場合)

$$T_o(v, \mathbf{p}) = min\{TG(v, \mathbf{p}) \cup TM(v, \mathbf{p})\} \tag{2}$$

where

$$TG(v, \mathbf{p}) = \{(v, u) \oplus T_o(u, \mathbf{p}) | v \in N(u)\} \tag{3}$$

$$TM(v, \mathbf{p}) = \{T_o(v, \mathbf{p_1}) \oplus T_o(v, \mathbf{p_2}) | (\mathbf{p_1} \cap \mathbf{p_2} = \phi) \wedge (\mathbf{p_1} \cup \mathbf{p_2} = \mathbf{p})\} \tag{4}$$

$\oplus$ は, 2 つの連結木（または, 1 つの連結木とエッジ）をマージして新しい連結木を作る演算子を表す．

式 (1) は, ノード v がキーワード部分集合 $\mathbf{p}$ を直接含んでいるとき, $T_o(v, \mathbf{p})$ を v とすることを表す．このとき T_o のコストは 0 となる．

式 (2) は, $TG(v, \mathbf{p})$ または $TM(v, \mathbf{p})$ に含まれる連結木のうち, 最小コストを持つ連結木が $T_o(v, \mathbf{p})$ となることを表す,

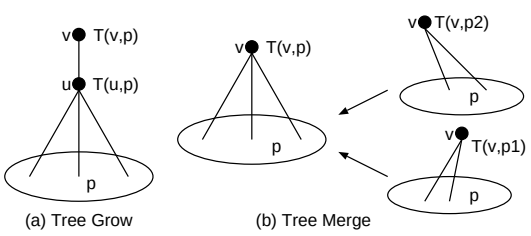

図 1　Tree Grow & Merge

式 (3) は図 1-(a) で示すように, ノード u をルートとする連結木から, ノード v へ grow することを表す (u, v を $V(G)$ の任意のノードとする)．

式 (4) は図 1-(b) で示すように, $(\mathbf{p_1} \cup \mathbf{p_2} = \mathbf{p}) \wedge (\mathbf{p_1} \cap \mathbf{p_2} = \phi)$ を満たす $\mathbf{p_1}, \mathbf{p_2}$ に対して $T(v, \mathbf{p_1})$ と $T(v, \mathbf{p_2})$ を merge して新しく $T(v, \mathbf{p})$ を作ることを表す ($\mathbf{p_1}, \mathbf{p_2}$ はキーワード集合 $\mathbf{P}$ の部分集合)．

3. DPBF の問題

3.1 問題点と解決手順

DPBF 法ではノートごとにコストの最も小さい連結木を保存するため, 最適な Top-1 解を求めることができる, しかし, 上位 2 から k 個までの解については最適ではない．DPBF-k の求める連結木は根付き木であり, 同型木という問題で上位 2 から k 個の解は実際に冗長解である可能性が高い．そのため, まず冗長解の問題を解決し, Top-k の解の最適化を次に扱う．

3.2 冗長解の定義

我々が考えた冗長解とすべきケースは三つある．一つはルートノード (根ノード) だけが異なってノード集合が全く同じ連結木である．これは DPBF-k の求める連結木が根付き木であるので, ルートノードさえ異なると, $|V(T)|$ 個の冗長な解が出力されうる (ここで $|V(T)|$ とは連結木 T に含まれる頂点の個数である)．

図 2-(a) では同型木上で各ノードをルートにすると同型木の解が多く出力される．例えばノード $v5$ をルートノートにして最適解を得る場合, ノード $v3$、$v4$ などをルートノードにする連結木が冗長解となる．もう一つは, 図 2-(b) で示すように, top-k の解となる連結木から成長させて得られる連結木である．

三つ目の冗長解は共通の経路を経由して, 全キーワードの部分集合 p を含む連結木を同じルートノードで merge することにより生成される連結木である．図 2-(c) で示すように, ノード u に grow してきた連結木 $T'(u, p1)$ と $T'(u, p2)$ を merge すると, top-k 解 $T(v, \mathbf{P})$ の冗長解 $T(u, \mathbf{P})$ が生成される．この解は grow 操作による冗長解よりコストが高い (エッジ (v, u) が重なってコスト 2 回カウントされる)．これは冗長解と見なし, 排除すべきである．

3.3 冗長解の回避方法

3.3.1 同型木の冗長解の回避法

同型木の排除については, 同型の連結木の数が多くないので, 解と確定される前, 既に解になった同じコストの連結木と木構造を比較すればすぐに排除できる．

3.3.2 解を成長させて得られる冗長解の回避法

DPBF では図 2-(b) の冗長解を回避できていない．図 3 で示すように, 連結木 $T'(v, \{p1\})$ と $T''(v, \{p2\})$ はノード w へ成長させて, さらにノード w で併合操作を行ったら連結木 $T_m(w, \{p1, p2\})$ が得られる．クエリーキーワード集合 $\mathbf{P}(=\{\mathrm{p1}, \mathrm{p2}\}$ に対しては確かに一つの解であるが, 図 3 で見ると, $T_m(w, \{p1, p2\})$ は最適解 $T(v, \{p1, p2\})$ の冗長解であるとよくわかる．DPBF ではこの冗長解を回避できていない．

この冗長解問題を解決するために, 最適解を成長させて図

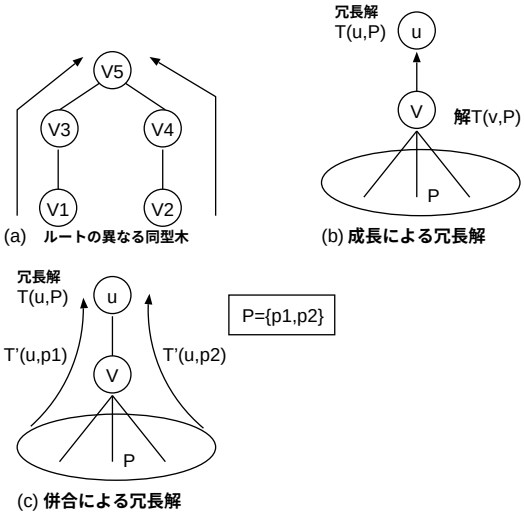

図 2　冗長解となるケース

2-(b) の冗長性を自動的に回避する方法を導入する．図 3 で説明すると，連結木 $T'(w,\{p1\})$ と $T''(w,\{p2\})$ がノード w で併合されて $T_m(w,\{p1,p2\})$ という候補解を探索キューに入れて確定を待つ．その後，コスト昇順の確定により解 $T(v,\{p1,p2\})$ が確定し，これをノード w へ成長させて冗長解というマークを付いた連結木 $T_g(w,\{p1,p2\})$ が候補としてキューに入る．merge 操作を行う際，共通エッジのコストが 2 回カウントされるので，$T_g(w)$ は $T_m(w)$ のコストより小さい，$T_m(w)$ は $T_g(w)$ に入れ換えられる．Top-k の解の計算では，コストの異なる解を出せば，正しい Top-k 解を見付けられる．連結木 $T_g(w)$ は冗長解マークが付いているので，探索処理には入るが，解としては避けられる．この操作を繰り返していくと，図 2-(b) と (c) のようなケースの冗長解はすべて回避できる．

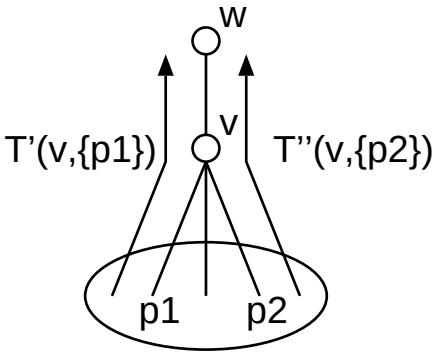

図 3　冗長解を自動的に抑制する

3.4　冗長抑制付き DPBF の欠点

以上の冗長抑制方針で冗長抑制付き DPBF を実装できた．そして ICDE 論文 [1] にある例題グラフで解の品質を検証する．

図 4 は ICDE の論文 [1] に使われているデータベースである．それをグラフ化して図 5 で示すようになる．ノードはタプルを，エッジは外部キー参照をそれぞれ表す.(エッジの重みを全部 1 にする) 冗長解抑制付き DPBF でクエリキーワード keyword, DB, Query, Jim によって top-3 を出した答えを図 6

Author

AID	Name
a1	Jim
a2	Robin

Citation

CID	Cite	Cited
c1	t1	t2
c2	t3	t2
c3	t5	t4
c4	t6	t7

Paper

AID	Title
t1	Keyword Search on RDBMS
t2	Steiner Problem in DB
t3	Efficient IR-Query over DB
t4	Online Cluster Problems
t5	Keyword Query over Web
t6	Query Optimization on DB
t7	Parameterized Complexity

Paper-Author

PID	AID
t2	a1
t4	a1
t3	a2
t4	a2
t5	a2
t6	a2
t7	a2

図 4　例題データベース [1]

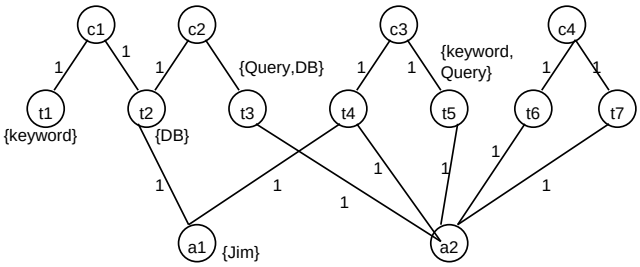

図 5　データグラフ

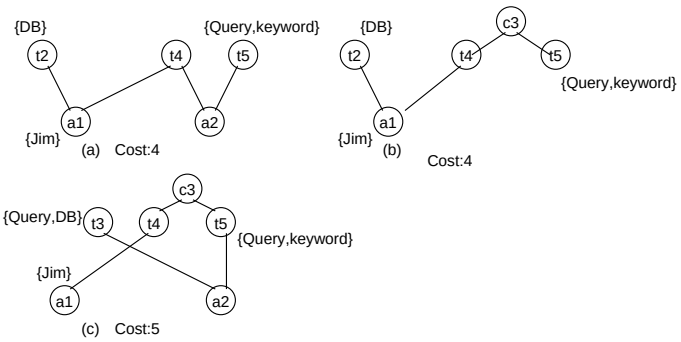

図 6　冗長抑制付き DPBF 法で出せる top-3 の解 (a)(b)(c)

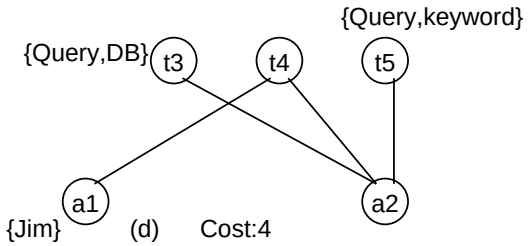

図 7　冗長抑制付き DPBF 法で出せない解 (d)

の (a),(b),(c) に示す．

しかし，図 7 の連結木 (d) は図 6 の連結木 (c) よりよい解と考えられるが，冗長抑制 DPBF では出せない解である．出せない原因は，ノートごとに連結木を一個しか覚えないため，解の連結木に含まれる潜在的なサブグラフが失われたからである．正確な top-k 解を求めることが必要であるので，我々はこの問題を解決するために MDPBF 法を [4] で提案した．

4.　MDPBF [4]

4.1　MDPBF の概要 [4]

正確な top-k を求めるためには，任意の $v,\mathbf{p}$ について，

$T_1(v,\mathbf{p}),\ldots,T_k(v,\mathbf{p})$ までを保存して, 計算するようにすればよい. そこで, 以下の式 (5)–(8) を計算するように, DPBF 法を改良する. 式 (5)–(8) は, それぞれ, 任意の $v,\mathbf{p}$ について, $T_1(v,\mathbf{p}),\ldots,T_k(v,\mathbf{p})$ までを保存しておくように, DPBF 法の式 (1)–(4) を改良したものである.

(v がキーワード $\mathbf{p}$ を直接含むとき)

$$T_1(v,\mathbf{p}) = v \tag{5}$$

(それ以外の場合, $1 \leqq n \leqq k$)

$$T_n(v,\mathbf{p}) = min\_n\ (\ TG(v,\mathbf{p}) \cup TM(v,\mathbf{p})\ ) \tag{6}$$

where

$$TG(v,\mathbf{p}) = \{\ (v,u) \oplus T_i(u,\mathbf{p}) \mid u \in N(v),\ 1 \leqq i \leqq k\ \} \tag{7}$$

$$TM(v,\mathbf{p}) = \{\ T_i(v,\mathbf{p_1}) \oplus T_j(v,\mathbf{p_2}) \mid \mathbf{p_1} \cap \mathbf{p_2} = \emptyset \wedge \mathbf{p_1} \cup \mathbf{p_2} = \mathbf{p},\ i \times j \leqq k\ \} \tag{8}$$

ここで, $T_n(v,\mathbf{p})$ は, v をルートとし, キーワード $\mathbf{p}$ を含む連結木の中で, n 番目に小さいコストを持つ連結木を表す. min_n は, 引数として与えられた連結木の集合から, n 番目にコストの小さい連結木を返す関数を表す. $N(v) = \{\ u \mid (v,u) \in E(G)\ \}$ は, v の隣接ノードの集合を表す.

式 (5) は, DPBF 法の式 (1) と同様に, ノード v がキーワード $\mathbf{p}$ を直接含んでいるとき, $T_1(v,\mathbf{p})$ を v (v のみからなる single node tree) とすることを表す. このとき, DPBF 法の場合と同様に, $T_1(v,\mathbf{p})$ のコストは 0 となる.

式 (6) は, DPBF 法の式 (2) を拡張したものであり, $TG(v,\mathbf{p})$ または $TM(v,\mathbf{p})$ に含まれる連結木のうち, n 番目 ($1 \leqq n \leqq k$) に小さいコストを持つ連結木が, $T_n(v,\mathbf{p})$ となるということを表す. ここで, $TG(v,\mathbf{p})$ は, ノード v の隣接ノード u をルートとする連結木 $T_i(u,\mathbf{p})$ ($1 \leqq i \leqq k$) から, ノード v へ grow した連結木の集合である (式 (7)). $TM(v,\mathbf{p})$ は, $(\mathbf{p_1} \cup \mathbf{p_2} = \mathbf{p}) \wedge (\mathbf{p_1} \cap \mathbf{p_2} = \emptyset)$ となる $\mathbf{p_1},\mathbf{p_2}$ に対して, $T_i(v,\mathbf{p_1})$ と $T_j(v,\mathbf{p_2})$ ($i \times j \leqq k$) を merge した連結木の集合である (式 (8)). 直観的に言えば, v の隣接ノード u をルートとする連結木 $T_i(u,\mathbf{p})$ ($1 \leqq i \leqq k$) から v へ grow したもの, または, $\mathbf{p_1} \cup \mathbf{p_2} = \mathbf{p}$ なる $\mathbf{p_1}$, $\mathbf{p_2}$ に対して, $T_i(v,\mathbf{p_1})$ と $T_j(v,\mathbf{p_2})$ ($i \times j \leqq k$) を merge したもの, のうち, n 番目 ($1 \leqq n \leqq k$) に小さいコストを持つ連結木が $T_n(v,\mathbf{p})$ となる, ということである.

ここで, 式 (8) の中の $i \times j \leqq k$ という条件について説明する. 今, $T_i(v,\mathbf{p_1}) \oplus T_j(v,\mathbf{p_2})$ よりもコストの小さい (または等しい) tree merge は, $T_i(v,\mathbf{p_1}) \oplus T_j(v,\mathbf{p_2})$ を含めて, 少なくとも, $|\{T_1(v,\mathbf{p_1}),T_2(v,\mathbf{p_1}),\ldots,T_i(v,\mathbf{p_1})\} \times \{T_1(v,\mathbf{p_2}),T_2(v,\mathbf{p_2}),\ldots,T_j(v,\mathbf{p_2})\}| = i \times j$ 個存在する. よって, $i \times j > k$ となる i,j の組み合わせについて, $T_i \oplus T_j$ は, top-k の中に入りえない. したがって, $i \times j \leqq k$ となる i,j の組み合わせについてのみ, tree merge を行うようにする.

式 (5)–(8) は, 任意の $v,\mathbf{p}$ について, 高々 top-k 個の連結木しか考慮しないで計算を行うが, $T_{k+1},T_{k+2},\ldots$ を部分木として持つ連結木は, それぞれの $v,\mathbf{p}$ について top-k 個の中に入りえない (GST-k の解になりえない) ので, $T_{k+1},T_{k+2},\ldots$ は top-k の計算 (GST-k の計算) に必要なく, 式 (5)–(8) によって, 確かに, GST-k の厳密解を求められる.

Algorithm 1: MDPBF 法

input : データグラフ G, キーワード全集合 $\mathbf{P}$, 出力する結果の数 k
output: $GST\text{-}k$ の解
variables:
Q_G : 連結木をコストの昇順で管理するプライオリティキュー .
$Q_L(v,\mathbf{p})$: $T_1(v,\mathbf{p}),\ldots,T_k(v,\mathbf{p})$ をコストの降順で管理するプライオリティキュー .

```
1  begin
2      Q_G ← ∅;
3      for each v ∈ V(G), p⊆P do Q_L(v,p) ← ∅;
4      for each v ∈ V(G) do
5          if v がキーワード部分集合 p を含む then
6              T_1(v,p) ← v;
7              T_1(v,p) を Q_G と Q_L(v,p) に enqueue;
8      num ← 0;
9      while Q_G ≠ ∅ do
10         Q_G から先頭要素を dequeue. これを T_i(v,p) とする;
11         if p = P then
12             output T_i(v,p);
13             num ← num + 1;
14         terminate if num = k;
15         for each u ∈ N(v) do
16             T(u,p) ← T_i(v,p) ⊕ (v,u);
17             if Q_L(u,p) のサイズが k より小さい then
18                 T(u,p) を Q_G と Q_L(u,p) に enqueue;
19             else if s(T(u,p)) < s(T_k(u,p)) then
20                 T_k(u,p) ← T(u,p);
21                 T_k(u,p) に関して Q_G と Q_L(u,p) を update;
22         p_1 ← p;
23         for each p_2 s.t. p_1 ∩ p_2 = ∅ do
24             for all j s.t. 1 ≦ j ≦ ⌊k/i⌋ do
25                 T(v, p_1 ∪ p_2) ← T_i(v,p_1) ⊕ T_j(v,p_2);
26                 if Q_L(v, p_1 ∪ p_2) のサイズが k より小さい then
27                     T(v, p_1 ∪ p_2) を Q_G と Q_L(v, p_1 ∪ p_2) に
                       enqueue;
28                 else if s(T(v, p_1 ∪ p_2)) < s(T_k(v, p_1 ∪ p_2))
                   then
29                     T_k(v, p_1 ∪ p_2) ← T(v, p_1 ∪ p_2);
30                     T_k(u, p_1 ∪ p_2) に関して Q_G と
                       Q_L(v, p_1 ∪ p_2) を update;
31 end
```

4.2 MDPBF 法の問題

1 節で述べたように, MDPBF 法 [4] では冗長列挙の抑制を考慮しなかった. そのため, MDPBF は DPBF [1] の冗長性を持っている.

5. 冗長抑制付き MDPBF

以下, MDPBF における冗長解の排除方法を述べる.

まず, 3 節の冗長解排除方法を Algorithm1(MDPBF 法) に適用する. そのうえでさらに, MDPBF 特有の冗長性を排除する.

5.1 MDPBF の冗長解と抑制方法

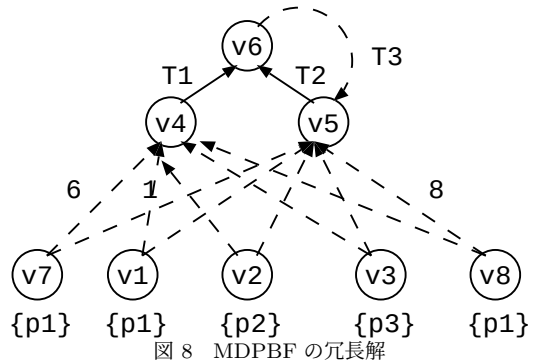

図 8　MDPBF の冗長解

MDPBF ではノードごとに k 個の連結木を保存するので, 同じ葉ノード集合から異なる経路で同じルートノードに辿り着いてきた連結木が存在する可能性がある, それを冗長解と見なして, 排除すべきである. 図 8 に示すように, 同じ葉ノード集合を持つ連結木 $T_1(v4, \{p1, p2, p3\})$ と $T_2(v5, \{p1, p2, p3\})$ はノード $v6$ に成長してきて, たとえコストが違うとしても, 連結木に含まれる情報 (キーワードを含む葉ノード) が全く同じなので、冗長の答えとする. これにより, ノード $v6$ から別の経路でノード $v5$ へ成長してきた連結木 T3 が $T(v5, \{p1, p2, p3\})$ と同じ葉ノードを持つので, top-10 を求めるとしても連結木 T3 を保存せずに済む, 以下, この冗長解抑制を入れた MDPBF 法を冗長抑制付き MDPBF と呼ぶ.

図 8 において, 冗長抑制付き MDPBF ではキーワード $p1$ を持つノード ($v1$, $v7$, $v8$) をそれぞれ葉ノードとする連結木は top-k の解として返せる. 冗長抑制付き DPBF ではノード $v1$ を葉ノードとするコスト最小の連結木 T($v4$,{p1,p2,p3}) しか返せない. なぜならノード $v4$、$v5$ でノード $v1$ から成長してきた連結木 $T(v4, p1)$ のコストが一番小さいので, ノード $v7$、$v8$ から成長してきた連結木 $T(v4, p1)$ が保存されない.

実装方法としては二番目以降の連結木をローカルキューに保存する際, 既に保存された連結木の葉ノード集合と比較する. 比較により同じ葉ノード集合の場合, 新しい連結木のコストが小さければ既に保存された連結木と入れ替える. そうでなければローカルキューに保存せず捨てる.

6. 時間計算量と記憶量

6.1 時間計算量

最悪場合の検索効率を評価しやすいため, 本研究ではバイナリヒープで冗長抑制付き MDPBF を実装した. top-k 個の解を見付けるまで grow 操作と merge 操作が繰り返される. ノードごとに k 個の候補連結木, 2^l のキーワード組合せがあるため, 全グラフデータベースにわたると, 併せて最大 $2^l nk$ 個の候補連結木がある. そのため, 候補連結木を保存し管理する GlobalQueue の最大サイズは $2^l nk$ 個があり, ループ(注1)の回る回数も最大 $2^l nk$ となる.

ループ全体において, 隣接ノードへの grow 操作は

$\sum_{v \in V(G))} |N(v)| k 2^l = 2^{l+1} mk$

回となる.(k:求める解の数, l:問い合わせキーワード数, m:グラフのエッジ数)

全く異なるキーワード集合を持つ連結木との merge 操作は

$nk \sum_{l_i=1}^{l} \binom{l}{l_i} 2^{l-l_i} = 3^l nk$

回となる.

以上により, バイナリヒープを使った場合冗長抑制付き MDPBF の時間計算量は:

$(2^{l+1} mk + 3^l nk) \log(2^l nk)$

となる。

l が十分小さい場合, 時間計算量は $(m+n)k \log(nk)$ となる.

6.2 記 憶 量

冗長抑制付き MDPBF では GlobalQueue と LocalQueue を共に使って連結木を保存し管理している.

GlobalQueue においては, 最悪の場合, グラフに含まれる全ての $T(v, \mathbf{p})$ を保存する必要があるので, MDPBF [4] と同じく $O(2^l nk)$ となる. また、LocalQueue において、GlobalQueue から dequeue されて確定された連結木 $T(v, \mathbf{p})$ と, まだ探索中の連結木を全て保存するので, 最悪の場合, GlobalQueue と同じく $O(2^l nk)$ となる.

以上により, 冗長抑制付き MDPBF の記憶量は GlobalQueue と LocalQueue の総和である. ここでは $O(2^{l+1} nk)$ となる.

7. 性 能 評 価

1000 個のノード、1500 個のエッジを持つランダムグラフを作り, そのランダムグラフにおいて, キーワード数 l を, 2 から 6 の間で変化させた場合の結果を図 9 と図 10 に示す. このとき, $k = 10, n = 1000$ とした. アルゴリズムは JAVA で実装され, CPU が 2.66GHz、メモリが 3GB の LinuxOS で性能テストを行った。

$l = 2$ の場合, 冗長抑制付き DPBF(以下 RDPBF と呼ぶ) と冗長抑制付き MDPBF(以下 RMDPBF と呼ぶ) アルゴリズムでは Loop 回数と実行時間がほとんど同じになる. $l = 4$ を越えると, RMDPBF のループ回数と実行時間が共に急増していく. $l = 6$ になった場合, RMDPBF のループ回数と実行時間が RDPBF の方より 10 倍ほどになることがわかる.

l を変化させた場合, RDPBF には 2^l 倍の計算量だけが増やすが, RMDPBF には $2^l k$ 倍の計算量が増やすため, ループ回数と実行時間が急速に増加していく.

また同じランダムグラフにおいて, 各アルゴリズム (RDPBF, RMDPBF) の Queue サイズ変化を図 11 で示す. クエリキーワード数 $l = 3$ として top-10 解を求める場合とした.

RDPBF の場合, RMDPBF より早めに Q_G ループの 3048 回目で探索処理を終わり, Queue の最大サイズは 2230 になる. 探索の処理時間は 286ms となる.

RMDPBF では RDPBF より多くループを回して (Q_G ループの 3601 回目で終わり), GlobalQueue の最大サイズは 6045 になり, LocalQueue の最大サイズは 9634 になる. 探索の処理時間は 621ms となる.

(注1)：Algorithm1 の 9 行目から 30 行目までの処理

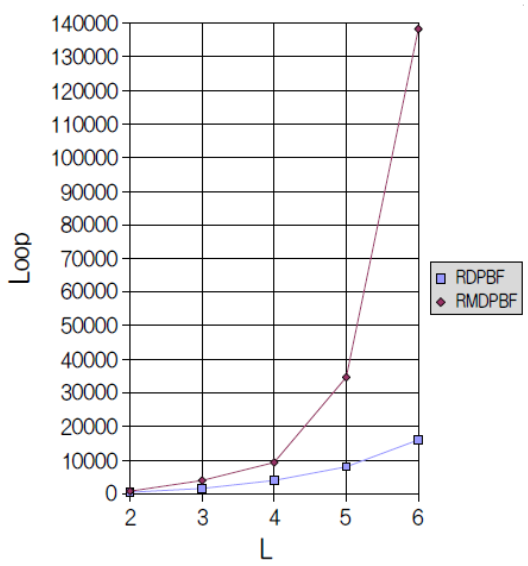

図 9　処理ループの回数 (キーワード数 l を変更する場合)

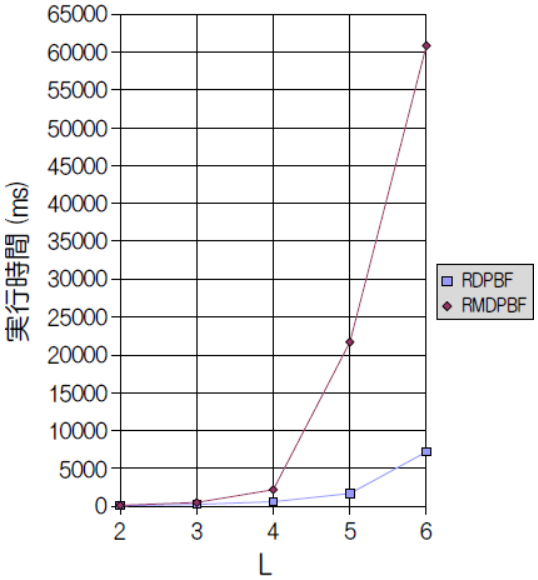

図 10　実行時間 (キーワード数 l を変更する場合)

RMDPBF で使ったメモリ領域は RDPBF より 8 倍ほどになる．そのうえ, GlobalQueue に含まれる連結木は候補として grow 操作と merge 操作を行う対象となるので, GlobalQueue のサイズが大きくなると, 探索のループ回数も増える．

merge 処理を行う際, MDPBF 法の $i \times j \leqq k$ という方法で merge の対象となる連結木の個数を k 個から k/i 個まで減らせることができるが, LocalQueue のサイズの増大に伴い, merge の対象となる連結木の個数は非常に増えていくため, RMDPBF 法は正確な top-k 解を求めるには高いコストをかけている．

8. ま と め

本稿では, グラフデータベースにおける正確な top-k 解の検索問題を扱い, シュタイナ木に基づいた従来手法 DPBF の改良として, 冗長抑制付き MDPBF 法を提案した．

本稿では, はじめに, 従来方式である DPBF における冗長解

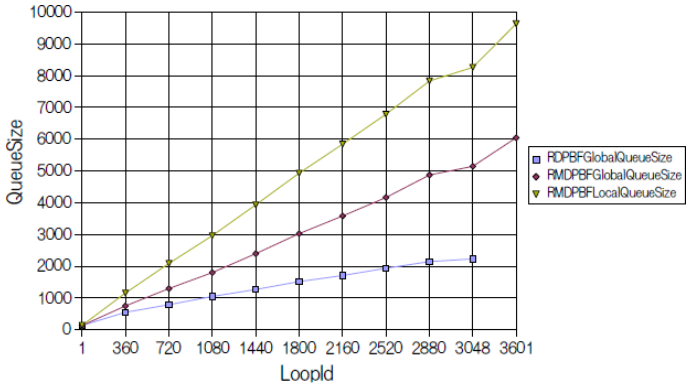

図 11　Queue サイズ変化 ($l = 3, k = 10$ の場合)

回避法を述べ, その上で, 簡単なグラフデータベース例を用いて, 冗長抑制つき DPBF では適切な答えを見落とすことを示し, 正確な top-k 解を出す必要性を述べた．次に, この問題を解決するため, 本稿では, DPBF の探索途中において各ノードに上位 k 個までの候補連結木を維持する戦略を入れた MDPBF 法を述べた．そして、MDPBF 法に特有な冗長解を述べ, その抑制方法を示した．

提案した冗長抑制つき MDPBF 法では, ノードごとに高々 k 個の候補木を保存するため, グローバルキューとローカルキューの記憶量と操作の手間が高くなる。具体的には, バイナリヒープを使う場合, $(2^{l+1}mk + 3^l nk)\log(2^l nk)$ の時間計算量と $O(2^{l+1}nk)$ の記憶領域が必要となるので MDPBF 法の記憶コストやループ回数は冗長抑制つき DPBF のそれらよりも最悪時に増える．従って, 冗長抑制付き MDPBF では, 10^6 ノード程度の巨大なグラフデータベースを扱う時には, 記憶量が大きくなりすぎて, 効率が下がることになる．この問題の解決が現在の課題である．

文　献

[1] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, X. Lin. Finding Top-k Min-Cost Connected Trees in Databases. In *Proc. of the 23rd International Conference on Data Engineering(ICDE'07)*, pp.836-845, 2007.

[2] S.E. Dreyfus, R.A. Wagner. The Steiner problem in graphs. *Networks*, 1:195-207, 1972.

[3] G. Bhalotia, A. Hugeri, C. Nakhe, S. Chakrabarti, S. Sudarshan. Keyword searching and browsing in databases using banks. *In Proc. of ICDE'02*, pp.431-440, 2002.

[4] 岡谷 昌史, 大森 匡, 星 守, グラフデータベースにおけるキーワード検索方式の改良. *FIT Forum '09*, A-020, 2009.

[5] V. Kacholia, S. Pandit, S. Charkrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional Expansion For Keyword Searches on Graphs. In *SIGMOD'07*, pp.305-316, 2007.

[6] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS Ranked Keyword Searches on Graphs. In *SIGMOD'07*, pp.305-316, 2007.

[7] 蒋 吏君, 大森 匡, 星 守, グラフデータベースにおける冗長解抑制を伴った正確な Top-k キーワード検索方式, 電気通信大学大学院情報システム学研究科 2009 年度修士卒業論文, 2010.