A-020

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

Modifying a Keyword Search Algorithm on Graph Databases

岡谷 昌史* 大森 匡* 星 守*
Masashi Okaya Tadashi Ohmori Mamoru Hoshi

1 はじめに

近年, グラフによって表現することのできるデータベースが増加している. 例えば, 関係データベースは, 関係データベースのタプルをノード, 外部キー参照をエッジと見なせば, グラフとしてモデル化される. そのため, 近年のデータベース研究分野では, データベースを表すグラフ (以下, グラフデータベースとか, データグラフと表記する) における効果的な検索方法が研究されている [1, 2, 3, 4]. 検索方式としては, top-k keyword search が一般的である. top-k keyword search とは, 検索対象のグラフに利用者が複数個のクエリキーワードを与えた時, 当該データグラフから全てのキーワードを含むような部分グラフを求めて, あるランキング基準によってランキングし, top-k(上位 k) 個の結果を返す方式である.

グラフデータベースに対する top-k keyword search アルゴリズムのうち, 代表的なものとしては DPBF 法 [1] がある. DPBF 法は, 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. ここで, 連結木とは, キーワードを直接含むノードを連結する木のことである. 連結木のコストとは, 連結木に含まれるエッジの重みの総和である. 全てのキーワードを含む最小コスト連結木を求める問題（すなわち, $k = 1$ の場合）は, minimum cost group Steiner tree problem （GST-1）として知られており, NP 困難な問題である [5]. したがって, 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問題（GST-k）も, 同様に, 非常に難しい問題となる.

DPBF 法の特徴は, クエリキーワード数が十分小さい場合には, GST-k の解を, $O(n \log n + m)$ (n：グラフのノード数, m：エッジ数) の計算量で求めることができることである [1]. しかし, DPBF 法では, 上位 1 番目の結果については厳密解を出せるが, 上位 2 番目以降の結果については近似解となる可能性がある. すなわち, DPBF 法では, GST-1 に対しては, 厳密解を出すことができるが, GST-k に対しては, 厳密解を出せるとは限らない. したがって, このままではユーザにとって重要な情報が失われてしまう可能性がある.

そこで本稿では, GST-k の厳密解を求めることができる新しいアルゴリズム「改良 DPBF 法」を提案する. 改良 DPBF 法の計算量は, クエリキーワード数と出力する結果の数の両方が十分小さい場合には, DPBF 法の場合と同様に, $O(n \log n + m)$ (n：グラフのノード数, m：エッジ数) となる.

以下, 2 節で関連研究を述べ, 3 節と 4 節で, DPBF の原論文 [1] に沿って, 扱う問題の定義と DPBF 法の概要を述べる. 5 節で, GST-k の厳密解を出すための我々の戦略を述べ, それに基づいて改良 DPBF 法を提案する. 6 節で, 簡単な冗長解抑制を行った場合も含め提案方法の実装評価を行い, 7 節でまとめる.

2 関連研究

グラフデータベースに対する top-k keyword search 手法のうち, DPBF 法以外の代表的な手法としては, BANKS-I [2], BANKS-II [3] がある. BANKS-I/II は, 全てのキーワードを含む top-k 個の最小コスト連結木 (正しくは, 1-star 木としてコストを計算する) を求める手法である. BANKS-I/II は, キーワードを直接含むノードから連結木のルートへの最短経路を結合することによって解となる連結木を得る. この方法では, 上位 1〜k 番目の全てにおいて, DPBF の解の考え方から言えば, 近似解となりうる. この他, 大規模なデータグラフを対象にインデックスとしてのデータ構造を導入して BANKS-II の改良法を高速化する手法 BLINKS[4] などがある.

3 問題定義

本節では, DPBF 法の原論文 [1] に基づいて, 本稿が対象とする問題の定義を行う.

3.1 データグラフについて

本稿では, 関係データベースのタプルをノード, タプル間の外部キー参照をエッジと見なしてグラフとする. このグラフをデータグラフと呼ぶ.

今, データグラフとして, 重み付きグラフ $G(V, E)$ を考える. ここで, V はノードの集合, E はエッジの集合である. エッジ $(u, v) \in E$ が非順序対（unordered pair）である場合（$(u, v) = (v, u)$）, G は無向グラフである. 簡単のため, 本稿では無向グラフのみを扱う. $N(v) = \{u \mid (v, u) \in E\}$ を $v \in V$ の隣接ノードの集合とする. グラフ $G(V, E)$ には重みがあり, 任意のエッジ $e \in E$ に対してエッジ重み $w_e(e)$ が存在するとする. $w_e(e)$ は, 非負であるとする. 以下, グラフ G に含まれるノードの集合を $V(G)$, エッジの集合を $E(G)$ と表す. グラフ G のノード数を $n (= |V(G)|)$, エッジ数を $m (= |E(G)|)$ とする. 本稿では, ノードの重みは仮定しない.

3.2 l-keyword query

l 個のキーワード $p_i (i = 1, \ldots, l)$ を与えたとき, データグラフ G に対する l-keyword query $\{p_1, p_2, \ldots, p_l\}$ を考える. データグラフのノードは関係データベースのタプルであるから, ある

* 電気通信大学大学院情報システム学研究科

ノードがキーワード p_i を満たす値を持つ場合がある．このとき，「ノード v はキーワード p_i を含む」と表記する．(または，3.3 で出てくる「木がキーワード集合を含む」という場合と明示的に区別するため，「ノード v はキーワード p_i を直接含む」とも書く)．

今，各 $i = 1,2,\ldots,l$ に対して，キーワード p_i を含むノードの集合 $V_i (\subseteq V(G))$ が存在する．($V_i \neq \phi$ とする)．よって，$V_1,\ldots,V_l$ までの，合計 l 個のグループが存在する．以下，V_i を，キーワード p_i のグループ，または，単にグループと呼ぶ．1 つのノードは複数のキーワードを含む可能性があり，ゆえに，1 つのノードは複数のグループに属する可能性がある．各グループは，転置インデックスのようなインデックスを用いれば，容易に求めることができる．

3.3 Minimum group Steiner tree problem (GST-1)

GST-1 とは，l-keyword query ($= \{p_1,p_2,\ldots,p_l\}$) が与えられたとき，データグラフ G から，各 i について $V(T) \cap V_i \neq \emptyset$ $(i = 1,2,\ldots,l)$ となるような最小コスト連結木 (minimum cost connected tree) を求める問題である．つまり、この連結木は，各 V_i に属すノードを少なくとも 1 つ以上持つ連結木でコスト最小のものである．

以下では，木 T を構成するノードに含まれるキーワードの (T の全ノードに渡っての) 和集合が $\mathbf{p} = \{p_{i_1},p_{i_2},\ldots,p_{i_k}\}$ となるとき，「木 T はキーワード集合 $\mathbf{p}$ を含む」という言い方をする．

GST-1 は，本質的には，minimum set cover problem と同等であり，NP 困難である [5]．本研究では，エッジの重要性は，その重みが小さいほど大きいと考え，簡単のため，連結木 T に対するコスト関数 s を以下で定義する．

$$s(T) = \sum_{e \in E(T)} w_e(e) \qquad (1)$$

したがって，最小コスト連結木を求める（GST-1 の解を求める）ということは，与えた l 個のキーワード全てを含む連結木のうち，最もノード間の関係が強いものを返すということである．

3.4 top-k group Steiner tree problem (GST-k)

GST-k とは，l-keyword query が与えられたとき，データグラフ G から，コスト関数 s によって，$s(T_1) \leq s(T_2) \leq \cdots \leq s(T_k)$ とランキングされるような，top-k 個の最小コスト連結木 $T_1,T_2,\ldots T_k$ を求める問題である．

3.5 本研究の対象とする問題

本研究が対象とする問題は，l-keyword query が与えられたときに，データグラフ G から GST-k の解を求めることである．

4 DPBF 法 [1]

本節では，DPBF 法の原論文 [1] に基づいて，DPBF 法の概要を説明する．

4.1 概要

DPBF 法 [1] は，任意のデータグラフから GST-k の解を求めるためのアルゴリズムである．上位 1 番目の解については厳密解となること，クエリキーワード数 l が十分小さく，定数とみなせる場合には，計算量が $O(n\log n+m)$ となること，が特徴である．

図 1 に，原論文 [1] で使われた DPBF 法の検索例を示す．図 1(b) は，関係データベース上のタプルをノード，外部キー参照を

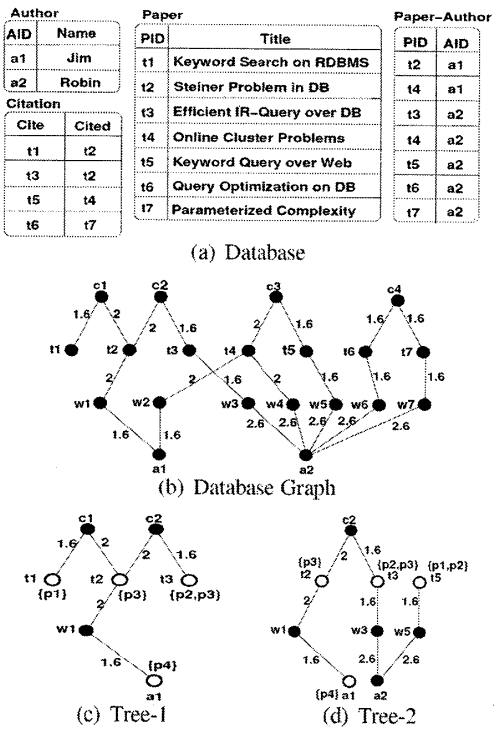


図 1　DPBF 法による検索例 [1]

エッジとみなして，図 1(a) の関係データベースをグラフ化したもの（データグラフ）である．(エッジのみ重さを持つ)．今，図 1(b) のグラフを，4 つのキーワード，`Keyword` (p_1), `Query` (p_2), `DB` (p_3), `Jim` (p_4) によって検索する場合を考える．図 1(c), (d) は，検索結果の候補となる，4 つのキーワードを全て含む，2 つの連結木を示している．図 1(c) の Tree-1 は，「`Jim` (a_1) が論文 t_2 を書き，その論文 t_2 が，2 つの論文 t_1, t_3 から参照されている」ことを表す．ここで，t_1 にはキーワード p_1, t_2 にはキーワード p_2, t_3 にはキーワード p_2,p_3 が含まれる．図 1(d) の Tree-2 は，「`Jim` (a_1) がキーワード p_3 を含む論文 t_2 を書き，その t_2 が，キーワード p_2, p_3 を含む論文 t_3 から参照されていて，t_3 の著者 a_2 が，キーワード p_1,p_2 を含む別の論文 t_5 を書いている」ことを表す．Tree-1(コスト 10.8) の方が Tree-2(コスト 15.6) よりキーワード間の関係を強く表す．DPBF は，このように，コストの小さい連結木を高ランクとして返す．

4.2 記号の定義

以下では，$\mathbf{P} = \{p_1,p_2,\ldots,p_l\}$ をキーワード全集合とする．また，記号 $\mathbf{p},\mathbf{p_1},\mathbf{p_2}$ を，$\mathbf{P}$ の (空でない) 部分集合を表す変数として用いる．ノード v をルート (=根ノード) とした木で，かつ，キーワード部分集合 $\mathbf{p} \subseteq \mathbf{P}$ を含むような任意の連結木を，$T(v,\mathbf{p})$ と表記する．$T(v,\mathbf{p})$ は 3.3 で決めたコストを持つ．v と $\mathbf{p}$ を決めたときに，ノード v をルートとし，かつ，キーワード部分集合 $\mathbf{p}$ を含む連結木の中で最小コストを持つ連結木を，$T_o(v,\mathbf{p})$ と表記する．

4.3 DPBF 法の考え方

DPBF 法では，以下の漸化式の計算を，キーワード全集合 $\mathbf{P}$ を含む連結木が k 個見つかるまで行うことによって，GST-k の解を求める．

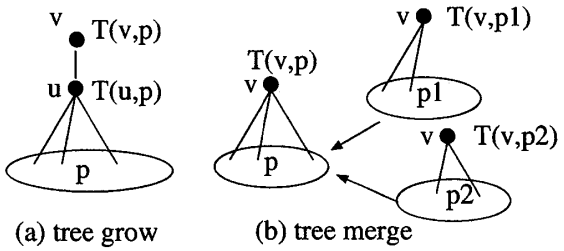


図 2　tree grow と tree merge [1]

(v がキーワード $\mathbf{p}$ を直接含むとき)

$$T_o(v, \mathbf{p}) = v \tag{2}$$

(それ以外の場合)

$$T_o(v, \mathbf{p}) = \min\ (\ TG(v, \mathbf{p}) \cup TM(v, \mathbf{p})\ ) \tag{3}$$

where

$$TG(v, \mathbf{p}) = \{\, (u, v) \oplus T_o(u, \mathbf{p}) \mid u \in N(v) \,\} \tag{4}$$

$$TM(v, \mathbf{p}) = \{\, T_o(v, \mathbf{p_1}) \oplus T_o(v, \mathbf{p_2}) \mid \mathbf{p_1} \cap \mathbf{p_2} = \emptyset \wedge \mathbf{p_1} \cup \mathbf{p_2} = \mathbf{p} \,\} \tag{5}$$

ここで, min は, 引数として与えられた連結木の集合から, 最小コストを持つ連結木を返す関数を表す. $\oplus$ 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 (4) の中の $(u, v) \oplus T_o(u, \mathbf{p})$ の計算を（tree) grow, 式 (5) の中の $T_o(v, \mathbf{p_1}) \oplus T_o(v, \mathbf{p_2})$ の計算を (tree) merge と呼ぶ.

木の grow と merge の説明を, 図 2 に示した. この図は, ある高さ $h-1$ の (根つき) 木 $T(u, \mathbf{p})$ をエッジ (u, v) によって高さ h の木 $T(v, \mathbf{p})$ へ grow する場合 (同図 (a)) と, 根ノード v を共有する木 $T(v, \mathbf{p_1})$ と $T(v, \mathbf{p_2})$ を merge して $T(v, \mathbf{p})$ (ただし, $\mathbf{p} = \mathbf{p_1} \cup \mathbf{p_2}$, かつ, $\mathbf{p_1} \cap \mathbf{p_2} = \emptyset$) を作る場合 (同図 (b)) を示す.

以上の前提の下で, 上述した各式は, 次の操作をする：

まず, 式 (2) 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（v のみからなる single node tree）とすることを表す. このとき, $T_o(v, \mathbf{p})$ のコストは 0 となる.

次に, 式 (3) 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 $T_o(u, \mathbf{p})$ から, ノード v へ grow した連結木の集合である (式 (4)). また, $TM(v, \mathbf{p})$ 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, $T_o(v, \mathbf{p_1})$ と $T_o(v, \mathbf{p_2})$ を merge した連結木の集合である（式 (5)）. 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 $T_o(u, \mathbf{p})$ 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, $T_o(v, \mathbf{p_1})$ と $T_o(v, \mathbf{p_2})$ を merge したもの, のうち, 最小コストを持つ連結木が $T_o(v, \mathbf{p})$ となる, ということである.

4.4 DPBF-1

まず最初に, GST-1 の解を求めるためのアルゴリズム DPBF-1 について説明する. DPBF-1 の擬似コードを図 3 に示す.

Algorithm 2 *DPBF-1*
input: database graph G, the set of keywords $\mathbf{P}$, and groups $V_1, \cdots, V_l$
output: *GST-1*

```
1: Let Q_T be a priority queue sorted in the increasing order of costs of trees;
2: Q_T ← ∅;
3: for each v ∈ V(G) do
4:   if v contains keywords p then
5:     enqueue T(v, p) = 0 into Q_T;
6: while Q_T ≠ ∅ do
7:   dequeue Q_T to T(v, p);
8:   return T(v, p) if p = P;
9:   for each u ∈ N(v) do
10:    if T(v, p) ⊕ (v, u) < T(u, p) then
11:      T(u, p) ← T(v, p) ⊕ (v, u);
12:      update Q_T with the new T(u, p);
13:  p_1 ← p;
14:  for each p_2 s.t. p_1 ∩ p_2 = ∅ do
15:    if T(v, p_1) ⊕ T(v, p_2) < T(v, p_1 ∪ p_2) then
16:      T(v, p_1 ∪ p_2) ← T(v, p_1) ⊕ T(v, p_2);
17:      update Q_T with the new T(v, p_1 ∪ p_2);
```

図 3　DPBF-1 の擬似コード [1]

DPBF-1 では, キーワードを直接含むノード (single node tree) (式 (2)) から始めて, tree grow/merge （式 (4), (5)) の 2 種類の操作によって連結木を構築してゆく. このとき, grow/merge によって作成された連結木を, プライオリティキュー Q_T によって, コストの昇順で管理する. 今, エッジ重みが非負であると仮定していることにより, tree grow/merge は, コストを減らす方向には働かないので, Q_T の top に存在する連結木は, 最適コストを持つことが保証される（ダイクストラの最短経路法と同じ戦略である. 図 4 を参照）. DPBF-1 では, この性質を利用して, （1）Q_T から連結木を dequeue して（式 (3) の計算に対応する）, （2）その連結木を grow/merge する, ということを, キーワード全集合 $\mathbf{P}$ を含む連結木が求まるまで, 繰り返すことによって, GST-1 の解を求める.

原論文 [1] の論述からはわかりにくいが, 上記の動作は, キーワードを直接含むノードを出発点として[*1], 連結木のコストの小さいもの優先で, グラフを探索することに等しい. (図 4 はこの探索の状態を表したもの). ただし, $T(v, \mathbf{p_1})$ によって v を訪問することと, $T(v, \mathbf{p_2})$ によって v を訪問することとは, v への相異なる訪問として扱われる. つまり, ノード v とキーワード部分集合 $\mathbf{p}$ の直積をノードとしたグラフを考えて, その中で最短経路を探索する.

上の考えに立つと, Q_T は, 訪問済みノードの隣接点で, まだ訪問していないノードの集合を管理する. Q_T からの dequeue は, 新たなノードの訪問 (と最小コストの決定) に対応する. tree grow は, 訪問したノードの隣接点を Q_T に enqueue する操作（グラフの探索）に対応する. tree merge は, 訪問済みノード $v(\mathbf{p_1})$ ($T(v, \mathbf{p_1})$ による v への訪問, という意味) から新たに訪問可能なノードとして $v(\mathbf{p_1} \cup \mathbf{p_2})$ ($T(v, \mathbf{p_1} \cup \mathbf{p_2})$ としての v への訪問) を作り, Q_T に enqueue する.

4.4.1 計算量/記憶量

DPBF-1 (図 3) の (時間) 計算量は, グラフのノード数 n, エッジ数 m, クエリキーワード数 l に対して, Q_T としてフィボナッチヒープを使って, $O(3^l n + 2^l((l + \log n)n + m))$ となる [1]. 故に, l

[*1] キーワードを直接含むノードは多数存在するので, 出発点が複数個あるとみなせばよい.

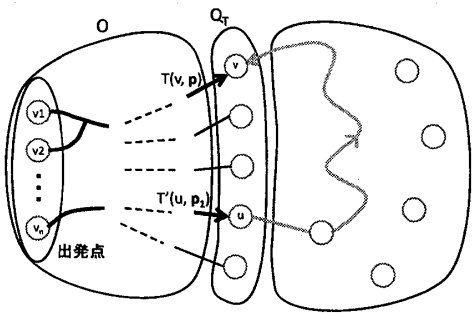


図 4　DPBF-1 の正当性

の値が十分小さく定数とみなせるなら，計算量は，$O(n \log n + m)$ となる．また，DPBF-1 の記憶量は，Q_T の最大サイズ，すなわち，$O(2^l n)$ である．

4.5　DPBF-k

GST-k の解を求めるアルゴリズム DPBF-k について説明する．DPBF-k は，DPBF-1 のアルゴリズムにおける処理ループ (図 3 の 6-17 行目) を，キーワード全集合 $\mathbf{P}$ 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 の解とする方法である [1]．すなわち，DPBF-k は，DPBF-1（図 3）の line 8 を，以下で置き換えたものである：

if $\mathbf{p} = \mathbf{P}$ **then**
　output $T(v, \mathbf{p})$; $i \leftarrow i + 1$;
terminate if $i = k$;

(i は 0 に初期化されるとする)．

DPBF-1 では，コストの小さいもの順で連結木を探索しているので，DPBF-k では，コストの昇順で，GST-k の解を求めることができる．DPBF-k における最悪の場合は，グラフに存在する全ての連結木を探索することになる場合であり，DPBF-1 における最悪の場合と等しい．よって，DPBF-k の計算量/記憶量は，DPBF-1 のそれに等しい．

5　提案手法：改良 DPBF 法

5.1　DPBF 法の問題点

$T_n(v, \mathbf{p})$ を，v をルートとし，キーワード部分集合 $\mathbf{p}$ 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とする．DPBF 法の用いる漸化式（4.3 節を参照）から容易にわかるように，DPBF 法は，任意の $v, \mathbf{p}$ に対して，$T_1(v, \mathbf{p})$ $(= T_o(v, \mathbf{p}))$ しか記録しない．したがって，ある $T_n(v, \mathbf{p})$ $(2 \leq n \leq k)$ を部分木として持つ連結木が，GST-k の解に属するとき（すなわち，GST-k の解に，ある $T_n(v, \mathbf{P})$ $(2 \leq n \leq k)$ が属するとき），DPBF 法はその連結木を解として出力できないので，近似解となる．

例えば，図 5 のような場合，DPBF 法は，T' は記録するが，T'' は記録しないので，T'' を部分木として持つ連結木が，GST-k の解に属する場合，その連結木を解として出力できないので，近似解となる．

この問題点が意味するのは，DPBF 法で検索を行った場合には，ユーザにとって重要な情報が見逃されてしまう可能性があるということであり，この問題点は，解決されるべき重要な問題点であると考えられる．以下，この解決法として，改良 DPBF 法を提案する．

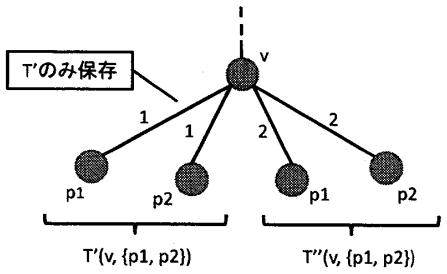


図 5　DPBF 法の問題点

5.2　DPBF 法の漸化式の改良

5.1 節で述べた問題点を解決するためには，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 (6)–(9) を計算するように，DPBF 法を改良する．式 (6)–(9) 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 (2)–(5) 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)

$$T_1(v, \mathbf{p}) = v \tag{6}$$

(それ以外の場合，$1 \leq n \leq k$)

$$T_n(v, \mathbf{p}) = \text{min\_n}\ (\ TG(v, \mathbf{p}) \cup TM(v, \mathbf{p})\ ) \tag{7}$$

where

$$TG(v, \mathbf{p}) = \{\ (v, u) \oplus T_i(u, \mathbf{p}) \mid u \in N(v),\ 1 \leq i \leq k\ \} \tag{8}$$

$$TM(v, \mathbf{p}) = \{ T_i(v, \mathbf{p_1}) \oplus T_j(v, \mathbf{p_2}) \mid \mathbf{p_1} \cap \mathbf{p_2} = \emptyset \wedge \mathbf{p_1} \cup \mathbf{p_2} = \mathbf{p},\ i \times j \leq k \} \tag{9}$$

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．$\oplus$ 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 (6) は，DPBF 法の式 (2) 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 (7) は，DPBF 法の式 (3) 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 $(1 \leq n \leq k)$ 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 $T_i(u, \mathbf{p})$ $(1 \leq i \leq k)$ から，ノード v へ grow した連結木の集合である（式 (8)）．$TM(v, \mathbf{p})$ 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 $T_j(v, \mathbf{p_2})$ $(i \times j \leq k)$ を merge した連結木の集合である（式 (9)）．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 $T_i(u, \mathbf{p})$ $(1 \leq i \leq k)$ 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 $T_j(v, \mathbf{p_2})$ $(i \times j \leq k)$ を merge し

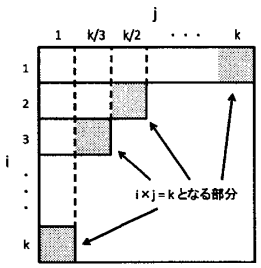


図 6　tree merge を行う範囲（改良 DPBF 法）

たもの，のうち，n 番目 $(1 \leq n \leq k)$ 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 (9) の中の $i \times j \leq k$ 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，$i \times j \leq k$ 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（図 6）．

式 (6)–(9) 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 (6)–(9) によって，確かに，GST-k の厳密解を求められる．

5.3　改良 DPBF 法の計算方法

改良 DPBF 法のアルゴリズムの擬似コードを Algorithm 1 に示す．

改良 DPBF 法では，DPBF 法と違って，$Q_T (= Q_G)$ には，任意の $v, \mathbf{p}$ について，コストの小さいものから順に最大 k 個の $T(v, \mathbf{p})$ を保存（$v(\mathbf{p})$ を k 個保存）するようにする（このとき，$Q_L(v, \mathbf{p})$ を利用する）．このとき，DPBF 法と同様に，キーワードを直接含むノードから，連結木のコストの小さいもの順で，探索（tree grow/merge）を行えば，連結木 $T(v, \mathbf{p})$ を k 回目に dequeue したとき（ノード $v(\mathbf{p})$ を k 回目に訪問したとき）には，その連結木 $T(v, \mathbf{p})$ が，$T_k(v, \mathbf{p})$ となる．すなわち，改良 DPBF 法では，DPBF 法と同様に，連結木のコストを priority とする priority-first search を行うことによって，式 (7) の計算を行う．GST-k の厳密解を求めるには，priority-first search（式 (7) の計算）を，キーワード全集合 $\mathbf{P}$ を含む連結木が k 個見つかるまで行えばよい（最初に見つけた k 個の連結木が GST-k の解となる）．直観的に言えば，改良 DPBF 法では，$T_k(v, \mathbf{p})$ の v を $v(\mathbf{p}, k)$ として，1 つの v を最大で $|2^{\mathbf{P}}| \times k$ 個に増やして考えて，DPBF 法を行うことによって，GST-k の厳密解を求める．

5.4　改良 DPBF 法が有効となる例

図 7(a) のデータグラフを，クエリ $\{p_1, p_2, p_3\}$ で検索する場合を考える．図 7(a) のグラフは，図 1(b) のグラフと同様に，論文とその著者の関係，論文間の引用関係を表したものである．

図 7(b) の連結木（コスト 15）は，上位 4 番目の厳密解であるが，DPBF 法では，図 7(b) の連結木を解として出力することができない．すなわち，著者 a_1, a_2，および，論文 t_3, t_4, t_5 が，キーワード $\{p_1, p_2, p_3\}$ と関係している，という情報が，DPBF 法による検索では，失われる．これは，（図 7(b) の連結木に含まれる）いかなるノードをルートとして考えても，図 7(b) の連結木は，（ルートとして考える，それぞれのノードについて）高々，上位 2 番目の連結木にしかならないことにより，DPBF 法では，図 7(b) の連結木を記録しておかないためである．一方，改良 DPBF 法では，任意の $v, \mathbf{p}$ について連結木を k 個まで保存するので，例えば，$k = 5$ とすれば，図 7(b) の連結木を上位 4 番目の解として出力できる．

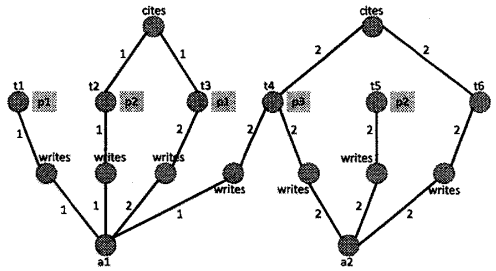
Algorithm 1: 改良 DPBF 法

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-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．

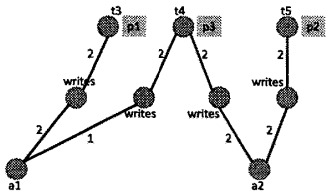
```
1  begin
2    Q_G ← ∅;
3    for each v ∈ V(G), p ⊆ P do Q_L(v, p) ← ∅;
4    for each v ∈ V(G) do
5      if v がキーワード部分集合 p を含む then
6        T_1(v, p) ← v;
7        T_1(v, p) を Q_G と Q_L(v, p) に enqueue;
8    num ← 0;
9    while Q_G ≠ ∅ do
10     Q_G から先頭要素を dequeue. これを T_i(v, p) とする;
11     if p = 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
```

5.5　改良 DPBF 法の計算量と記憶量

改良 DPBF 法（Algorithm 1）の計算量は，グラフのノード数 n, エッジ数 m, クエリキーワード数 l, 出力する結果の数 k に対して，$O(3^l nk \log^2 k + 2^l k((l + \log k + \log n)n + m \log k))$ となる [6]．l と k の値が十分小さく，定数とみなせるならば，計算量は，$O(n \log n + m)$ となる．l と k の値が十分小さい，という条件は，実用においては，容易に満たされると考えられる．また，改良 DPBF 法の記憶量は，キュー全ての最大サイズ分，つまり，



(a) 検索するデータグラフ



(b) DPBF 法では出力されない解

図 7　改良 DPBF 法が有効となる例

$O(2^l nk)$ となる.

ここで, 改良 DPBF 法では, 任意の $v, \mathbf{p}$ に対して, 連結木を k 個まで保存する. また, 改良 DPBF 法では, DPBF 法と違って, ローカルキュー Q_L を使用する. そのため, 改良 DPBF 法の計算量/記憶量は, DPBF 法の計算量/記憶量に, k（1 つの $v, \mathbf{p}$ に対して保存する連結木の数）, または, $\log k$ （Q_L に対する操作の手間）, を掛けた形になっている.

6　予備的評価

6.1　概要

実験には, DPBF 法, 改良 DPBF 法, DPBF 法と改良 DPBF 法のそれぞれに簡単な冗長解抑制を加えたもの, の 4 つのアルゴリズムを使用する. 簡単な冗長解抑制とは, 次の (i), (ii) の両方を DPBF と改良 DPBF の各々に加えたものである：

(i) top-k として指定された値 k について, 相異なるコストを持つ k 個の連結木を解として出力するように修正する. – この理由は, top-k 解の計算に際して, DPBF 法と改良 DPBF 法が, 解となる木 T を一つ見つけた後で, T とルート (根ノード) だけ異なる木を, 別の解として出力しうるからである. (この性質を, 冗長性 1 と呼んでおく). これは, DPBF 法および改良 DPBF 法の求める連結木が, 解として本来必要な自由木ではなく, 根付き木であるためである*2. (実装では, 同一コストの解のうち最初の解を出力して, それ以後のそのコストの解は, 内部処理自体は普通に行うが, 解としては出力せず, コストの異なる解を k 個得るまで計算を行うとした).

(ii) キーワード全集合を含む連結木がキュー Q_T(または Q_G) から取り出されて見つかった時に, その木をそれ以上 grow しない. – これは, 他の解となる木を grow しただけの木を top-k の解として出力することをできるだけ抑制するためである. (ただし, これを行っても, 「よりコストの小さい解を真に含む木を解とすること」(冗長性 2 と呼ぶ) を完全に回避できるわけではない.)

以下, 上記の冗長解抑制方法 (i)(ii) を共に行う場合の DPBF 法と改良 DPBF 法を, 各々, DPBF*, 及び, MDPBF* と表記する. (i)(ii) 共に行わない場合の DPBF 法と改良 DPBF 法を, 各々, DPBF, MDPBF と表記する.

DPBF*, MDPBF*は, 各々, 対応する DPBF, MDPF よりも, 解を求めるまでのループ回数 (DPBF なら図 3 の while ループ, MDPBF なら Algorithm2 の 9 行目の while ループのこと) は多くなるが, 冗長性 1 または 2 を満たす解のうち自明なもので上位 k 個になることをできるだけ避け, より実質的な top-k 探索を行う場合を意図している*3. ただし, この処理をしても DPBF*, MDPBF*共に, 冗長解を完全に回避できるわけではないし, 本来 top-k 内にいるべき解を得られない可能性がある. それでも, 5.1, 5.4 節で述べた top-k 解を計算対象に入れている点は, DPBF* にはない MDPBF*の特徴である.

実験は, CPU 3.4GHz, メモリ 3GB, Linux OS の PC で行った. アルゴリズムは Java で実装し, 各キュー (Q_T または Q_G) として二分ヒープを使った.

6.2　実験結果

DPBF, DPBF*, MDPBF, MDPBF* の 4 つのアルゴリズムを実行し, それぞれの結果を比較した. 実験では, 実行時間とループ回数 (DPBF/DPBF*なら図 3 の while ループ, MDPBF/MDPBF*なら Algorithm2 の 9 行目の while ループのこと. Q_T または Q_G から dequeue して最小コストを持つ木を決定する回数を表す) の 2 つを性能指標として計測した.

まず, キーワード数 $l = 3$, 出力する結果の数 $k = 5$ としたときに, グラフのノード数 n を, n = 10K, 50K, 100K, 200K, 300K ($K = 10^3$) と変化させた場合の性能を評価した. 各アルゴリズムを実行するときには, その都度, サイズ n のグラフを人工的にランダムに作成し, その後, その作成したグラフに対してアルゴリズムを実行する, という方式をとった. 各 n に対して, 各アルゴリズムを 10 回実行し, 各指標の 10 回の平均をとった. ランダムに作成するグラフのエッジ数 m は, $m = 3n$ とした. それぞれのエッジには, ランダムに, 1〜100 の間の重みを割り当てた. 各グループ V_i $(1 \le i \le l)$ （キーワード p_i を含むノードの集合）のサイズは, $|V_i| = n/100$ とした. 各グループ V_i は, ランダムに作成した（ランダムにノードを割り当てた）.

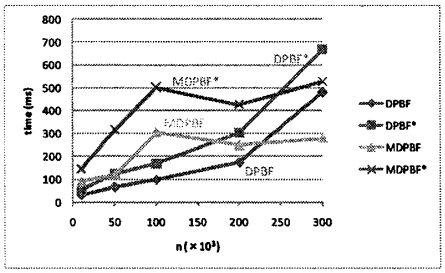
続いて, $n = 10$K, $k = 5$ としたときに, キーワード数 l を, 2〜6 の間で変化させた場合, および, $n = 100$K, $l = 3$ としたときに, 出力する結果の数 k を, k = 1, 5, 10, 15, 20 と変化させた場合, の性能も同様に評価した.

■n を変化させた場合　グラフのノード数 n を, n = 10K, 50K, 100K, 200K, 300K と変化させた場合の結果を図 8 に示す. このとき, $l = 3$, $k = 5$ とした.

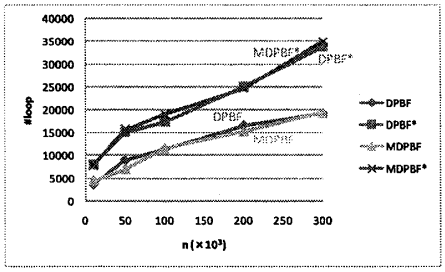
DPBF, DPBF*では, n に対して, 実行時間, ループ回数共にほぼ線形に変化している. 一方, MDPBF, MDPBF*では, ループ回数はほぼ線形に変化しているが, 実行時間は, 不規則に変化

*2 故に, 解として連結木 T が出力された時, 最大で $|V(T)|$ 個の冗長な解（ルートが異なるのみで形は同じ連結木）が同一コストの解として出力されうる.

*3 DPBF の原論文 [1] は冗長性 1,2 の扱い方を述べていない.

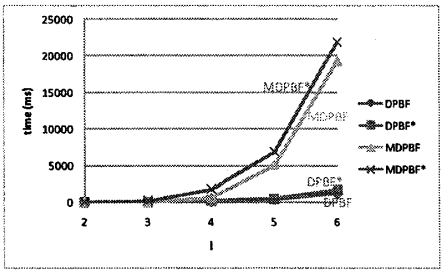


(a) 実行時間

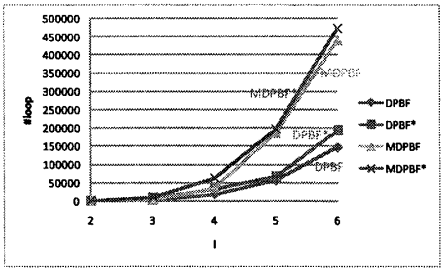


(b) ループ回数

図 8　n を変化させた場合の結果



(a) 実行時間



(b) ループ回数

図 9　l を変化させた場合の結果

している．これは，MDPBF, MDPBF*では，1 ループあたりの計算時間が，グラフのノード数によって大きく変化するということを表す．このことは，改良 DPBF 法の，任意の $v, \mathbf{p}$ について連結木を k 個まで保存するという性質により，MDPBF, MDPBF* では，キューに入れられる連結木の数（キューのサイズ）が，グラフの状態（ノード数）によって，大きく変化するためであると考えられる（1 ループあたりの計算量については，dequeue の手間が支配的であることに注意）．また，冗長解抑制ありのものは，冗長解抑制なしのものよりも，実行時間，ループ回数共に大きくなっている．冗長解抑制をした場合，ループ回数は約 1.5 倍以上増えていることから，冗長解抑制を行わない場合 (DPBF と MDPBF) では指定 $k = 5$ 個の解の中に同一コストの解を多く含むことがわかる．つまり，6.1 の (i) で指摘した冗長性 1 となる解を出力して終了する場合が多いことがわかる．

■ l を変化させた場合　次に，キーワード数 l を，2〜6 の間で変化させた場合の結果を図 9 に示す．このとき，$n = 10\text{K}$, $k = 5$ とした．

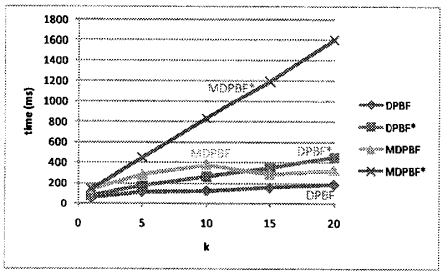
4 つのアルゴリズム全てにおいて，実行時間，ループ回数共に，l に対して指数関数的に変化している（最悪計算量の式と一致する）．ループ回数における DPBF*(または，DPBF) と MDPBF*(または，MDPBF) の間の違いは，約 2.5 倍であるが，実行時間における DPBF*(または，DPBF) と MDPBF*(または，MDPBF) の間の違いは，約 10 倍となっている．このことより，MDPBF*(または，MDPF) の 1 ループあたりの計算量は，DPBF*(または DPBF) のそれよりも大きいことがわかる．これは，改良 DPBF 法では，（1）任意の $v, \mathbf{p}$ について連結木を k 個まで保存することにより，キューのサイズが，DPBF 法よりも平均的に大きい，（2）DPBF 法とは違って，ローカルキューの操作も行う，の 2 点が影響していると考えられる．一方，ループ回数に関しては，図 8(b) で l, k を固定して n を変化させた場合には DPBF と MDPBF(または，DPBF*と MDPBF*) がほぼ同様の振る舞いをしていたが，図 9(b) で n, k を固定して l(つまり，計算したい木の複雑さ) を変化させた場合，DPBF と DPBF*, MDPBF と MDPBF*が，それぞれほぼ同様の振る舞いをし，MDPBF*(または MDPBF) が DPBF*(または DPBF) より大幅にループ回数を増やす ($l = 5$ では 4 倍)．これは，l を増やすと，n を変化させた場合と比べて，解となる連結木のサイズが大きくなり（解を見つけるまでに，たどらなければならない経路が長くなり），それによって，改良 DPBF 法と DPBF 法の間のループ回数（および，実行時間）に，差が明示的に現れる（最悪計算量の式に近づく）ためと考えられる．

■ k を変化させた場合　最後に，n, l を固定して，出力する結果の数 k を，$k = 1, 5, 10, 15, 20$ と変化させた場合の結果を図 10 に示す．$n = 100\text{K}$, $l = 3$ とした．

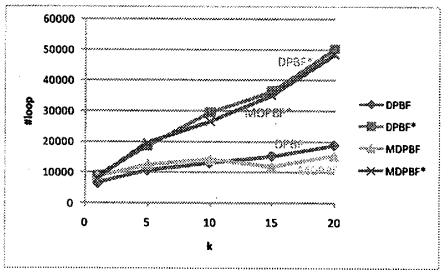
4 つのアルゴリズム全てにおいて，k に対して，実行時間，ループ回数共に，ほぼ線形に変化している．

DPBF と MDPBF のループ回数，DPBF*と MDPBF*のループ回数は，それぞれ，ほぼ等しいが，計算時間については，DPBF*と MDPBF*の差は，DPBF と MDPBF のそれよりもかなり大きい．これは，今回の冗長性削除法の下では，ループ回数が少し大きくなっただけで（すなわち，解となる連結木のサイズが少し大きくなっただけで），改良 DPBF 法と DPBF 法の間での，キューのサイズ（キューに入れられる連結木の数）の違いが，たちに大きくなり，実行時間に差が出てくる（すなわち，任意の $v, \mathbf{p}$ について連結木を k 個まで保存することが，大きく影響してくる）ためと考えられる．

以上の結果をまとめると，今回の簡単な冗長解抑制法の下でも，改良 DPBF 法の計算時間・ループ回数は，DPBF 法のそれらと比べ，解となる連結木のサイズが少しでも大きくなると（解を見つけるまでに，たどらなければならない経路が，少しでも長



(a) 実行時間



(b) ループ回数

図 10　k を変化させた場合の結果

くなると），大きく増加することがわかる．特に，キーワード数 l の増加や top-k 解の数 k の増加に対しては，ローカルキューの記憶量と操作の手間が大きく働いて，改良 DPBF 法は元の DPBF 法よりも大きく計算コストが増える．しかし，6.1 節 (i)(ii) の意味での冗長解をできるだけ避けつつ，5.1, 5.4 節で述べた top-k 解を計算対象に入れて動作できる点は MDPBF*の特徴である．

7　まとめ

本稿では，GST-k の厳密解を求めるための新しいアルゴリズム「改良 DPBF 法」を提案した．改良 DPBF 法の計算量は，グラフのノード数 n，エッジ数 m，クエリキーワード数 l，出力する結果の数 k に対して，$O(3^l nk\log^2 k + 2^l k((l+\log k+\log n)n + m\log k))$ となる．l と k の値が十分小さく，定数とみなせるならば，計算量は，$O(n\log n + m)$ となる．また，必要な記録量は，$O(2^l nk)$ である．改良 DPBF 法は，DPBF 法が原理的に無視する解 (の候補となる木) を top-k 解の計算の対象に入れている点が特徴である．一方，DPBF 法と改良 DPBF 法も共に，top-k 解探索では，根ノードが異なるのみでノード集合とエッジ集合が同じ木を異なる解として数える，解となる木を真に含む解を数える，という性質がある．試作評価では，こうした冗長解の出力をできるだけ避けるため，コストの異なる解を k 個見つけるまで計算するように修正した場合も含めて，DPBF 法と改良 DPBF 法を試作評価した．その結果，改良 DPBF 法は，現実的な計算時間で top-k 解を計算できたが，一方で，キーワード数 l の増加や top-k 解の数 k の増加に対しては，ローカルキューの記憶量と操作の手間が高く，DPBF 法よりも処理コストが大きく増大することも示された．本問題におけるデータベース検索上の妥当な冗長解の定義と改良 DPBF 法での効率的な冗長解回避方法，及び，実データ上の解の品質 (情報としての良さ) の評価が現在の課題である．

参考文献

[1] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k Min-Cost Connected Trees in Databases. In *ICDE'07*, pp.836–845, 2007.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE'02*, pp.431–440, 2002.

[3] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. In *VLDB'05*, pp.505–516, 2005.

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

[5] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(1):195–207, 1972.

[6] 岡谷，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，電気通信大学大学院情報システム学研究科修士論文，2009 年 3 月．