日本データベース学会和文論文誌
Vol. 20-J, Article No. 16, 2022 年 3 月

木分解の圧縮および解集合プログラミングに基づく問合せ処理法の提案と評価

小島 和之 [1]　関 浩之 [2]

グラフの木分解は, 大規模グラフデータの問合せに対して有効なアプローチであるが, 木分解自体には大きな計算量を要する. そこで本研究は木分解を再利用するために木文法に基づく圧縮手法を提案する. 併せて問合せ処理について解凍することなく実行する手法を提案する. 問合せ時に使用するアルゴリズムについて直接記述する方法と解集合プログラムを利用する方法を提供する. 実験の結果, 十分な圧縮率が得られること, 問合せを直接記述する方法においては解凍せずに問合せ処理を行う方法が従来法より計算効率がよいこと, 解集合プログラムを利用する方法では記述容易性・了解性が向上することがわかった.

1　まえがき

グラフ構造は科学データなど実世界の大規模データを表現するために用いられる. グラフに対する問題で効率のよいアルゴリズムが知られていないものが多数ある. こうした問題の計算量を削減するために, 木分解と呼ばれる手法を用いて, 与えられたグラフから木構造 (分解木とよぶ) を構成し, 分解木に対して動的計画法を用いて元のグラフに対する問題を解くというアプローチが知られている. ここで分解木の頂点は元のグラフの部分グラフに対応しており, それらの部分グラフの頂点の最大数から 1 を引いた数をその分解木の幅とよぶ. 幅がある定数以下の分解木のクラスに対しては, ある種の NP 完全な問題をこのようなアプローチによって元のグラフのサイズの多項式時間で解くことができる.

一方, 木分解の計算にはグラフ全体の構造情報を必要とし計算コストが高い. また分解木に含まれる部分グラフには重複が多いためデータサイズが元のグラフに比べ大きく, データの圧縮を行えることが望ましい. しかし, 分解木に対して単純な圧縮アルゴリズムを適用した場合では構造情報が損なわれてしまう. そのため分解木に対する問合せや更新時には一旦解凍してから処理を行わなければならず非効率的である.

本研究では, 与えられたグラフの分解木に対して圧縮法を効果的に適用可能な表現形式および, 圧縮された分解木に対して解凍せずに直接的に問合せを行う方法として, 問合せアルゴリズムを直接記述する方法 [25] と解集合プログラムの形式で記述する方法 [26] を提案する. この 2 つの記述法を用いて問合せを記述し評価実験を行った. 筆者らが圧縮法として利用している TreeRePair [20] は, 入力として与えられた木において隣接する頂点のラベル対のうち最頻出するものを非終端記号へ置換することを繰り返し, 直線的文脈自由木文法に変換する. TreeRePair は木文法に基づく多くの圧縮法の中で圧縮率や圧縮に要する時間の点で優れている. 更に直線的文脈自由木文法び対して解凍することなく問合せや更新を行う手法について研究がなされている [16].

以下, 本論文では諸定義を行った後, 後続の圧縮効果が得やすい分解木の表現形式 (3 節), 木文法を用いた圧縮法 (4 節) を提案する. 次に, 圧縮データへの (解凍を伴わない) 直接問合せ法を例を用いて説明する (5 節). 最後に分解木に対して (木圧縮を意識せずに) 動的計画法による問合せ処理の記述を支援する解集合プログラムの枠組みを提案する (6 節). 解集合プログラムは規則の集合として記述された論理プログラムである. プログラムの解は任意の規則を満たす原子命題の集合 (解集合) であり, 解集合の探索にはソルバを利用する. 解くべき問題の定義を規則の集合として与えると, そのままプログラムとして実行できるという利点がある. 5 節と 6 節では実装システムを用いて提案手法を評価した結果を述べる. 2 つの問合せ手法をいくつかのベンチマークに対して実行した結果, (1) 十分な圧縮効果が得られたこと, (2) 手作業で記述した問合せについては直接問合せが多くの場合で効率よく実行できたこと, (3) 解集合プログラミングを用いることにより宣言的な問合せ記述ができ, 問合せプログラムの了解性が向上したが, 今回の実装では直接問合せの実行効率には改善の余地があることがわかった.

関連研究　グラフ圧縮には, [19] 等の文字列圧縮を応用した伝統的な方法があるが, 近年, 簡潔データ構造を用いた圧縮 [6,7,13] や文法を用いた圧縮 [22] などのように, 圧縮データを解凍することなく近傍走査が行えるグラフ圧縮法が主流となっている. k^2-tree [7] では, グラフを隣接行列で表し, その行列を適切な区間で区切ったものを簡潔な表現に符号化することで圧縮を行なっており, 疎なグラフに対して効果的に圧縮できる. k^2-tree はその汎用性が高く, 他の圧縮法の内部で使用されている [9, 17, 22]. 特に [17] では, [8] を拡張したものと k^2-tree を組み合わせた手法が提案されており, Web グラフの圧縮では 0.9 ~ 1.5bpe という高い圧縮率を示している. 他の手法として, 圧縮グラフに対する問合せをあらかじめ限定し, その問合せに不必要な情報を削ることで効果的に圧縮を行う手法も存在する [12, 23, 24]. [14] では, 隣接リストを用いて圧縮された Web グラフに対するページランクの効率的な計算方法が示されているが, Web グラフのスパース性と局所性を利用しているため, 一般のグラフに対してはあまり効果的でない. グラフ文法を用いた gRePair [22] では, 圧縮グラフ上で 2 頂点間の到達可能性を圧縮データサイズの線形時間オーダで判定できることを証明しているが, 実験的評価は行われていない. グラフ圧縮と直接問合せ法を概観するには [3, 21] を参照されたい.

木分解を利用して木幅が有界なグラフに対する問題を効率良く解く手法を解集合プログラミングに導入した先駆的研究として [18] がある. 問合せアルゴリズムを解集合プログラムによって記述することで多くのグラフ問題を扱っている. この研究に基づき, 木分解を前提とした解集合プログラミングを支援する統合環境 D-FLAT が開発された [2, 4]. 6 節の研究方針および実装システムも, D-FLAT の設計方針を参考にしている. ただし, 本研究で

[1] 学生会員　名古屋大学 情報学研究科
k.kojima@sqlab.jp
[2] 正会員　名古屋大学 情報学研究科
seki@sqlab.jp

は分解木をさらに木文法に圧縮することを前提としており, ユーザが圧縮後のデータの細部に依存せずに解集合プログラミングを行える点に特長がある.

2 定義

2.1 グラフ

無向グラフは 2 項組 $G = (V, E)$ である. ここで V は頂点の有限集合, E は辺すなわち頂点の非順序対 (u, v) $(u, v \in V)$ の有限集合である. 以下では無向グラフを単にグラフとよぶ.

グラフ $G = (V, E)$ における独立集合 $I \subseteq V$ とは任意の異なる頂点間に辺が存在しない頂点の部分集合である. つまり任意の $u, v \in I$ $(u \neq v)$ について $(u, v) \notin E$.

グラフ $G = (V, E)$ における 3 彩色とは写像 $\sigma : V \to C$ $(C = \{red, blue, green\})$ である. ただし任意の辺 $(u, v) \in E$ について $\sigma(u) \neq \sigma(v)$.

2.2 木分解

グラフ $G = (V, E)$ に対して次の条件を満たす $\mathcal{T}$ を木分解により G から得られた木または分解木とよぶ [11]. 分解木 $\mathcal{T}$ の各頂点 x (以降, $x \in \mathcal{T}$ と表記する) には G の部分グラフ $B_x = (V_x, E_x)$ が割当てられる. これを x のバッグとよぶ. バッグは以下の条件を満たさなければならない:

- $\cup_{x \in \mathcal{T}} V_x = V$.
- G の全ての辺 $(u, v) \in E$ について $(u, v) \in E_x$ となる $x \in \mathcal{T}$ が存在する.
- $x, y, z \in \mathcal{T}$ とする. 頂点 y が x から z への $\mathcal{T}$ における経路上にあるとき $E_x \cap E_z \subseteq E_y$.

2.3 木および木文法

ランク付きアルファベット Σ に対し, $a \in \Sigma$ のランク (引数の数) を $\mathrm{rank}(a)$ で表す. Σ と変数の集合 $\mathcal{X}$ によって生成される木全体の集合を $T(\Sigma, \mathcal{X})$ と表記する. 木 $t \in T(\Sigma, \mathcal{X})$ において各変数 $x \in \mathcal{X}$ が高々 1 回しか現れないとき t は線形であるという. 木を 2 分木へ変換する方法として fcns 符号化 [10] が知られている. $t_1, \ldots, t_n$ を木とし, $t_1 = f(s_1, \ldots, s_m)$ とする. fcns 符号化 $\mathrm{fcns}(t_1, \ldots, t_n)$ を次のように定義する.

- $\mathrm{fcns}() = \#$.
- $\mathrm{fcns}(t_1, \ldots, t_n) = f(\mathrm{fcns}(s_1, \ldots, s_m), \mathrm{fcns}(t_2, \ldots, t_n))$.

$Gr = (N, \Sigma, R, S)$ を文脈自由木文法 (CFTG) とよぶ [20]. ここで N, Σ $(N \cap \Sigma = \emptyset)$ はランク付きアルファベットで, それぞれ, 非終端記号, 終端記号の有限集合, R は生成規則の有限集合. 生成規則は $A(x_1, \ldots, x_n) \to t$ の形をしたものであり, $A \in N$, $\mathrm{rank}(A) = n$, $t \in T(\Sigma \cup N, \{x_1 \ldots x_n\})$. $S \in N$ は開始非終端記号で, $\mathrm{rank}(S) = 0$.

木 t における変数 $x_1, \ldots, x_n$ をそれぞれ木 $t_1, \ldots, t_n$ に置き換えて得られる木を $t[x_1/t_1, \ldots, x_n/t_n]$ と表す. $s = C[z/A(t_1, \ldots, t_n)]$, $s' = C[z/t[x_1/t_1, \ldots, x_n/t_n]]$ であるような $A(x_1, \ldots, x_n) \to t \in R$ と $C \in T(\Sigma \cup N, \mathcal{X} \cup \{z\})$ が存在するとき, $s \Rightarrow_{Gr} s'$ とかく. $\Rightarrow_{Gr}$ の推移閉包, 反射推移閉包をそれぞれ $\Rightarrow^{+}_{Gr}$,$\Rightarrow^{*}_{Gr}$ とかく. また, Gr から生成される木言語を $L(Gr) = \{t \mid S \Rightarrow^{+}_{Gr} t$ かつ $t \in T(\Sigma, \emptyset)\}$ と定義する.

直線的文脈自由木文法 (SLCFTG) は以下の条件を満たす CFTG $Gr = (N, \Sigma, R, S)$ のことをいう.

(1) 各非終端記号 $A \in N$ について, ちょうど 1 つの生成規則 $A(x_1, \ldots, x_n) \to t \in R$ が存在し, かつ t が線形である.
(2) 各非終端記号 $A \in N$ について, $A(x_1, \ldots, x_n) \Rightarrow^{+}_{Gr} s$ であるならば, s に A が出現しない.

$A(x_1, \ldots, x_n) \to t \in R$ について, t を t_A と書き, また t_A を A の右辺とよぶ. 任意の SLCFTG Gr について, 開始非終端記号 S からの導出 $S \Rightarrow^{+}_{Gr} s$ は (1) より決定的に行われ, (2) よりどの規則も再帰的に適用されないので, $L(Gr)$ の要素数は 1 である.

TreeRePair [20] は入力として与えられた木を次の方法で SLCFTG に変換する圧縮法である. 頂点 u の i 番目の子頂点を v とする. 頂点 u, v のラベルをそれぞれ f, h としたとき, これらの頂点が構成するダイグラムを (f, i, h) と表す. TreeRePair はダイグラムのうち最頻出するものを非終端記号に置換することを繰り返して木文法を構成する. ただし置換はその前後でサイズが減少する場合のみ行う. このようにして木から SLCFTG を構成することを圧縮という. 逆に SLCFTG Gr から $t \in L(G_r)$ である木 t を求めることを解凍という.

2.4 解集合プログラム

At を原子命題 (アトムと略す) の集合とする. At 上の解集合プログラム Π とは次の形式の規則 r の有限集合である. $r = a_1 \vee a_2 \vee \ldots \vee a_k \leftarrow b_1, \ldots b_m, not\ b_{m+1}, \ldots, not\ b_n$ $(a_1, \ldots, a_k, b_1, \ldots, b_n \in At)$

$r \in \Pi$ の構成要素を, $h(r) = \{a_1, \ldots, a_k\}$, $b^+(r) = \{b_1, \ldots, b_m\}$, $b^-(r) = \{b_{m+1}, \ldots, b_n\}$ と定義する. $b^+(r) = b^-(r) = \emptyset$ ならば r をファクトとよぶ. ファクトである規則 r については $\leftarrow$ を省略して $a_1 \vee \ldots \vee a_k$ と表記する.

アトムの集合 I と規則 r に対し, $I \cap h(r) \neq \emptyset$ か $b^-(r) \cap I \neq \emptyset$ か $b^+(r) \backslash I \neq \emptyset$ のいずれかが成り立つとき, すなわち, I が $b^+(r)$ のアトムを全て含み $b^-(r)$ のアトムをどれも含まないならば左辺のアトムを少なくとも 1 つ含むとき, I は r を満たすという. 特に $k = 0$ すなわち左辺が空 (false) であるとき, 規則 r は右辺のすべてが同時には成り立たないことを表している. I が Π に属する任意の規則を満足するとき Π のモデルとよぶ. I が $\Pi^I = \{h(r) \leftarrow b^+(r) \mid r \in \Pi, b^-(r) \cap I = \emptyset\}$ の (集合の包含関係に関する) 極小モデルであるとき, I は Π の解集合であるという.

3 分解木の表現形式

本節では分解木を表現し, かつ効果的に圧縮可能なデータ構造 (以降, この段落では表現とよぶ) を提案する. 本研究において, 分解木は木分解ツールの出力を利用する. 木の根頂点の選択や兄弟頂点の並べ方は複数考えられるが, 圧縮はこれらに依存しない. 一方, 分解木上の問合せ処理は木の構造によって効率が変化することが考えられるが, これは木分解に基づくグラフアルゴリズム共通の問題であるため, 本研究では検討の対象としていない.

分解木は木構造を持つためそれ自体にも木圧縮のアルゴリズムを適用できる. しかし分解木の頂点の内容であるバッグはグラフ

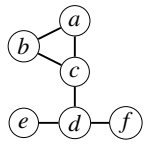

図 1　グラフ G

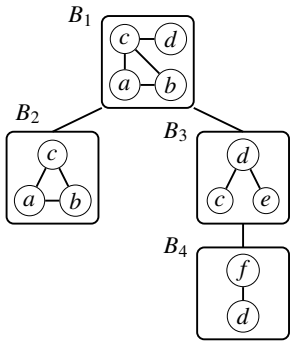

図 2　分解木

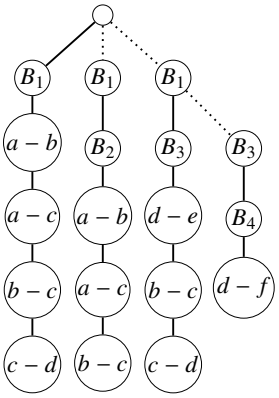

図 3　提案する分解木の表現

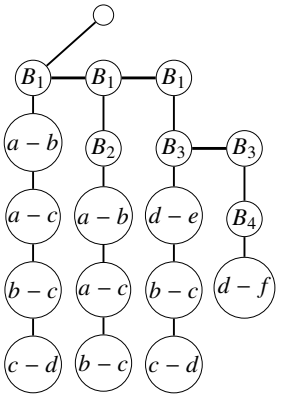

図 4　fcns 符号化を行った分解木

構造であるためバッグ自体を頂点のラベル名とみなした場合, 異なる頂点 x と y のバッグ B_x と B_y に大きな重複があっても木圧縮ではそれを考慮して圧縮を行うことはできない．そこでバッグについてはそれに属する各辺を頂点で表しそれらの頂点を辺で直列に接続し経路 (分岐のない木) として表現する．与えられた分解木 $\mathcal{T}$ に対し, 次の手順で圧縮に適した $\mathcal{T}$ の表現形式 $\mathcal{D}$ を生成する ($\mathcal{D}$ も木である).

1. $\mathcal{T}$ の各頂点 x をラベルにもつ頂点を $\mathcal{D}$ に含める．これをバッグ頂点とよぶ.
2. 各バッグ頂点 v(そのラベルを x とする) について, 以下を行う.
   $\mathcal{T}$ における x の親 y をラベルにもつ頂点 u を v の親として $\mathcal{D}$ に含める. (ただし x が $\mathcal{T}$ の根頂点であるときは u のラベルは未定義とし, u を $\mathcal{D}$ の根とする.)
3. $\mathcal{D}$ の頂点のうち, ラベルが同じ頂点すべてが兄弟頂点となるように辺を追加する. ただしバッグ頂点が最も左の子頂点となるように配置する.
4. 各バッグ頂点のラベルである部分グラフについて, その各辺をラベルにもつ頂点 (辺頂点) を $\mathcal{D}$ に含める．これらの辺頂点をバッグ頂点 u の子として接続する. 辺頂点の接続する際の順序はラベルである辺について, その 2 つの端点のうちより出現するバッグ数が少ない端点に注目し, この頂点について昇順とする．端点の出現数が等しい場合は, もう一方の端点の出現数を用いて順序を決定する.

上記の 4. において辺頂点を出現数に従って並べている．これによってバッグ間に共通する辺が分解木上で隣り合う可能性が高まるため圧縮の効果が高まることを予備実験を確認している. 図 2 の分解木をこの方法で表現したものを図 3 に示す．以降, この表現も分解木とよび, あるバッグ B_x をラベルにもつ頂点をバッグ頂点, また, ある辺 (u, v)(図 3–5 では, $u-v$ と表している) をラベルに持つ頂点を辺頂点とよぶ.

4　木文法を用いた圧縮法

本研究では木圧縮の手法である TreeRePair を利用し, その前提の上で分解木の圧縮および非解凍問合せを行う手法を提案している．圧縮および非解凍問合せの効果を発揮できるよう, 分解木のデータ構造を提案した点が本研究における新規性である．なお, バッグに含まれるグラフにグラフ構造を対象とした圧縮を適用する方法も考えられるが, バッグ内のグラフは密となる (頂点数に対して辺数が多い) 傾向があり, そのようなグラフに対する効果的圧縮は期待できない. そのため本研究ではデータ構造を木構造に統合することで圧縮および問合せ処理を単純化することを狙いとしている.

4.1　分解木の fcns 符号化

本研究で木圧縮に用いる TreeRePair はまず, 入力された分解木の fcns 符号化を行う．図 3 の分解木を fcns 符号化したものを図 4 に示す．図 3 における破線は fcns 符号化によって削除される辺, 図 4 における太い実線は追加される辺を表す．符号化された分解木には以下の特徴がある.

1. 根頂点の子はただ一つに限られる.
2. 同じバッグ頂点が複数個存在するとき, それらの位置関係は兄弟関係から祖先子孫関係に変わる.
3. 符号化前にバッグ頂点 B_x が B_y の子頂点であったならば符号化後もそうである.
4. 同一バッグ内の辺頂点間の位置関係も符号化によって変化しない.

4.2　分解木から得られる SLCFTG

前節で述べた fcns 符号化後の分解木は続いて SLCFTG に変換される．分解木において非終端記号に置換されるダイグラムはバッグ頂点同士, 辺頂点同士に限られる．その理由は, バッグ頂点と辺頂点の組み合わせからなるダイグラムは, 同一のバッグ B_x に対しては 1 回しか分解木に現れないことによる.

Example 4.1 (ダイグラムの非終端記号への置換例) 図 *4* に示した分解木に対するダイグラムの置換について説明する.

- ダイグラム $(B_1, 1, (a, b))$,$(B_2, 1, (a, b))$,$(B_3, 1, (d, e))$,$(B_4, 1, (d, f))$ は一度のみ出現するため非終端記号に置換されることはない.
- ダイグラム $((a, b), 1, (a, c))$ は *2* 回出現するため非終端記号 A_1 への置換が行われる．このときダイグラムを生成する規則 $A_1 \to ((a, b), 1, (a, c))$ が構成される.

- ダイグラム $((b,c),1,(c,d))$ は *2* 回出現するため非終端記号 A_2 への置換が行われる．このときダイグラムを生成する規則 $A_2 \to ((b,c),1,((c,d),1,\alpha))$ が構成される．

以上より, 図 *4* に TreeRePair を適用すると図 5 に示した SLCFTG が得られる．

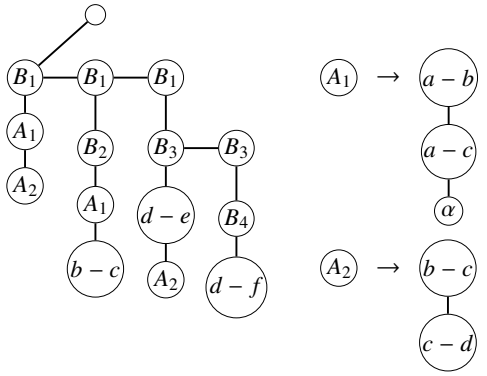

図 5　図 4 の分解木から得られる SLCFTG

5　直接問合せ例:最大独立集合問題

グラフ問題の例として最大独立集合問題を取り上げる．独立集合はグラフの頂点集合であり, 任意の 2 頂点間に辺が存在しないものである．最大独立集合問題は, 与えられたグラフの独立集合のうち頂点数が最大であるものを求める問題である．この問題は NP 完全であるが幅有界な分解木に変換できるグラフの部分クラスに対しては多項式時間可解であることが知られている．

分解木における問合せ処理は以下の手順で計算できる．まずそれぞれのバッグ (すなわち元のグラフの部分グラフ) に対する全ての局所解を求める (5.2 参照)．次にバッグが分解木の葉でない場合は，自身の局所解に対して子頂点に対応するバッグの局所解との組み合わせを求める．この組合せ操作を分解木においてボトムアップに繰り返し，根頂点のバッグにおいてコスト (最大独立集合問題では頂点数) が最大である集合が元のグラフにおける解となる．なお，このアルゴリズムの時間計算量は分解木の幅の指数オーダとなる．(グラフの大きさに対して幅が十分小さい定数以下である場合に，効率よく動作することを目的としている．)

5.1　単一バッグ内の独立集合の計算

バッグ B_x 内の独立集合を求める単純なアルゴリズムについて述べる．このアルゴリズムはバッグ B_x を入力し B_x の独立集合のすべてからなる集まり $\mathscr{I}_x$ を出力する．

アルゴリズムは辺 (u,v) を一つずつ読み込み，その端点を独立集合に追加する操作を繰り返し，部分グラフにおける独立集合を求める．辺 (u,v) を読み込んだとき端点 u を含まない既存の独立集合 I_x に u を追加した集合 $I_x \cup \{u\}$ を $\mathscr{I}_x$ に追加する．v についても同様の処理を行う．一方, u,v を含む独立集合は（辺 (u,v) が存在することが分かったので）$\mathscr{I}_x$ から削除する．これを E_x の全ての辺に対して繰り返すことで独立集合が求められる．

5.2　SLCFTG の特徴を用いた単一バッグ内の独立集合の計算

木文法において非終端記号の右辺は複数のバッグに出現する部分グラフを表す (4.2 参照)．非終端記号の右辺が表す部分グラフに対する独立集合を予め計算し，それ以外の部分，つまりその非終端記号を参照している部分についての独立集合と合成することで実行時間の短縮が見込まれる．

このアルゴリズムの詳細を図 6 に示す．B_x について非終端記号の右辺から計算された独立集合の集まり $\mathscr{I}_y$ とその非終端記号を参照している部分に対する独立集合の集まり $\mathscr{I}_x$ が入力である．まず独立集合 $I_x \in \mathscr{I}_x$ が全ての $I_y \in \mathscr{I}_y$ と交差する場合, I_x を $\mathscr{I}_x$ から削除する．次に $\mathscr{I}_x$ に含まれない $\mathscr{I}_y$ の要素である独立集合を追加する．更に追加した集合と $\mathscr{I}_x$ に最初から存在する集合の和集合を追加する．

Input: $V_x, V_y, \mathscr{I}_x, \mathscr{I}_y$
Output: $\mathscr{I}_x$
foreach $I_x \in \mathscr{I}_x$ **do**
  $match \leftarrow false$
  foreach $I_y \in \mathscr{I}_y$ **do**
    if $I_x \cap (V_y - I_y) = \emptyset$ **then**
      $match \leftarrow true$
  if $match = false$ **then**
    $\mathscr{I}_x \leftarrow \mathscr{I}_x - \{I_x\}$
foreach $I_y \in \mathscr{I}_y$ **do**
  foreach $I_x \in \mathscr{I}_x$ **do**
    if $((V_x - I_x) \cap I_y) \cap ((V_y - I_y) \cap I_x) = \emptyset$ **then**
      $\mathscr{I}_x \leftarrow \mathscr{I}_x \cup \{I_y\} \cup \{I_x \cup I_y\}$
return $\mathscr{I}_x$

図 6　非終端記号の右辺から計算された独立集合の合成アルゴリズム

5.3　分解木全体の最大独立集合の計算

分解木の各バッグの独立集合に対して子バッグの独立集合との和集合を取ったときも独立集合となる組合せで要素数最大となるものを求める．子バッグが複数存在する場合についてもそれぞれについて独立に組み合わせを求める．これを分解木においてボトムアップに繰り返す．元のグラフにおける最大独立集合は分解木の根頂点にあるバッグにおける最大独立集合と一致する．

5.4　評価実験

5.4.1　実装ツール・実験環境・ベンチマーク

グラフに対する木分解の計算ツール htd [1] に分解木を提案形式 (3 節) で出力する処理を追加した．また，htd で提供されている木分解の計算法のうち Minimum fill ordering アルゴリズムを使用した．評価実験に使用したデータセット [5] は表 1 の通りである．これらのグラフに対する木分解によって得られた分解木の規模は表 2 の通りである．

表 1　グラフデータ

#	ファイル名	頂点数	辺数
1	myciel5	47	236
2	huck	74	301
3	queen5	25	160
4	david	87	406

表 2　分解木データ

#	頂点数	幅
1	26	21
2	34	10
3	7	18
4	59	13

5.4.2　圧縮率

TreeRePair によって表 2 の分解木を圧縮した結果を表 3 に示す．表 3 における圧縮率は，圧縮後の木文法のデータサイズを圧縮前の分解木のデータサイズで割った値を表している．

表 3　分解木の圧縮結果

ファイル名	分解木 (kB)	木文法 (kB)	圧縮率 (%)
myciel5	8.22	1.47	17.9
huck	9.34	2.10	22.4
queen5	7.91	1.06	13.3
david	22.44	3.32	14.8

5.4.3　問合せの実行効率

問合せ例として取り上げる最大独立集合の頂点数はそれぞれ#1 は 23 個，#2 は 27 個，#3 は 5 個，#4 は 36 個であった．表 4，表 5 は表 2 に示した分解木についてそれぞれ最大独立集合の要素数算出を 100 回行ったときの平均実行時間，最大使用メモリを示したものである．これらの表において直接問合せの欄の「入力処理」は終端記号，非終端記号に付与されたラベルおよび木文法の読み込みである．「最大独立集合計算」はバッグに含まれる頂点，辺の読み込みと最大独立集合の要素数の計算を指す．

■5.4.3.1　**実行時間**　解凍にかかる時間が全体の実行時間に占める割合が小さいため，解凍による影響はない．ただし入力の処理にかかる時間に着目すると直接問合せを行った場合が実行時間が短い．

最大独立集合の計算にかかる時間を比較する．直接問合せでは図 6 のアルゴリズムを利用することで単一バッグ内の独立集合を計算する際の実行時間を解凍後問合せより短縮することができた．

■5.4.3.2　**必要メモリ量**　直接問合せではいずれのグラフデータに対しても入力処理に必要なメモリ量が大きい．今回使用している SLCFTG は部分的な解凍が可能である．しかし直接問合せ時においても全ての頂点ラベルを処理開始時に読み込むことから重複が少ないグラフデータについては直接問合せの恩恵が得にくい．

最大独立集合の計算では#1，#3 については直接問合せが解凍後問合せより必要なメモリ量が小さい．実行時間と同様に直接問合せについては図 6 のアルゴリズムが利用でき重複部分に対する独立集合の計算を減らすことができるためである．対して#2，#4 については分解木の重複部分が少ないため，独立集合を格納するメモリ量の減少量が処理関数に必要なメモリ量に満たない．そのため必要メモリ量が増大する．

表 4　問合せ時間

ファイル名	直接問合せ		解凍後問合せ	
	入力処理 (*ms*)	最大独立集合計算 (*ms*)	解凍 (*ms*)	最大独立集合計算 (*ms*)
myciel5	0.913	139.619	1.4203	440.018
huck	1.060	0.573	1.876	0.687
queen5	0.603	67.733	1.182	462.364
david	1.5172	2.339	3.392	3.413

表 5　問合せ時の使用メモリ

ファイル名	直接問合せ		解凍後問合せ	
	入力処理 (kB)	最大独立集合計算 (kB)	解凍 (kB)	最大独立集合計算 (kB)
myciel5	4188	11008	4068	11272
huck	4116	4208	4332	3472
queen5	4120	5700	4000	6008
david	4140	4744	4168	3776

6　解集合プログラム

6.1　解集合プログラムを使用した問合せ

5 節では圧縮された分解木を解凍することなく問合せを行う方法を提案した．しかし, 問合せアルゴリズムは問題ごとに個別に記述する必要があった．例えば最大独立集合問題の問合せを記述するために C++ で約 180 行を要し, 圧縮データへ直接問合せを行うためプログラムの了解性も低かった．

本節では解集合プログラムによってアルゴリズムを宣言的に記述しソルバで実行することで問合せの記述コストを低減することを提案する．解集合プログラムを利用した問合せの記述は D-FLAT においても提案されている．本研究では, 分解木に基づくプログラミングを支援する基本的な機能は D-FLAT を参考にして設計したが, 圧縮箇所の直接問合せを支援する機能を新規に考案 · 実装し (6.4, 6.5), その効果を実験的に評価した (6.6)．グラフ, 分解木, 解集合プログラムについての処理フローを図 7 に示す．しかし, 分解木の情報を解集合プログラムのソルバに入力することは容易でない．さらに, 異なる問合せ間で共有できる処理が相当数存在する．そこで分解木のバッグに含まれる部分グラフの情報を参照するための解集合プログラムの組込み述語を実装した．分解木に対する問合せはバッグごとの局所解の計算とそれらの合成プロセスからなる．これら各プロセスに必要な処理についても実装した (図 7 の実行エンジン)．本研究では解集合プログラムの実行には Clingo [15] を使用している．

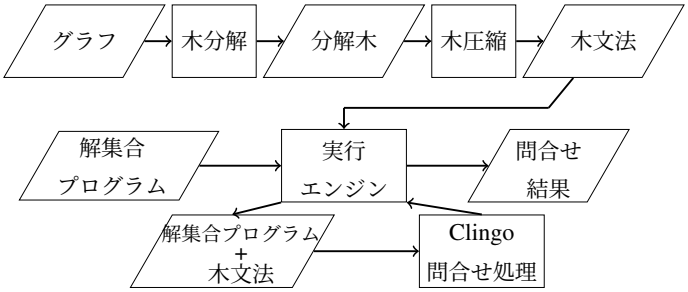

図 7　本提案手法による処理フロー

6.2　Clingo における解集合プログラム

Clingo 処理系は規則左辺が成り立つとき後の推論のため記憶するので, これを「ファクトを生成する」とよぶ. 与えられたグラフ $G = (V, E)$ の独立集合を求める解集合プログラムを示す.

$$\{in(X) : vertex(X)\}. \tag{1}$$
$$\leftarrow in(X), in(Y), edge(X,Y). \tag{2}$$
$$cost(C) \leftarrow \#count\{X : in(X)\}. \tag{3}$$

規則に現れる $vertex(X)$ は $X \in V$, $edge(X,Y)$ は $(X,Y) \in E$ を表す. $in(X)$ は X が独立集合に属することを表す. $cost(C)$ はここでは頂点数を表す.

規則 (1) は Clingo 独自の記法 (以降, Clingo 記法とよぶ) であり, $vertex(X)$ を満たす X が非決定的に選ばれ, ファクト $in(X)$ が生成される [15]. (これに対してもし, 規則を $in(X) \leftarrow vertex(X)$. とした場合 $vertex(X)$ であるすべての X について $in(X)$ が生成される.) 規則 (2) は $in(X), in(Y)$ である任意の X, Y について $edge(X,Y)$ が成り立たないことを表す (規則左辺が空であることに注意. 2.4 参照). つまり独立集合が X, Y を要素として持つとき, その頂点間に辺が存在しないことを表す (2.1 参照). 規則 (3) は $in(X)$ が成立しているような X の個数を数え上げて C に代入し $cost(C)$ を生成する. $\#count\{X : P(X)\}$ は $P(X)$ を満たす X の個数を表す Clingo 記法である.

6.3　分解木における問合せ

問合せアルゴリズムの設計を 3 色問題を例にして述べる. 分解木に対するグラフ問題の問合せは各バッグにおける局所解を計算し, それらを木に沿って合成することで行う. 問合せアルゴリズムとして与える解集合プログラムもこの各段階ごとに設計する.

まず局所解の計算処理について述べる. 規則 (4) の左辺は Clingo 記法であり, X に対してただ 1 つの色 (C) を与え, ファクト $map(X,C)$ を生成する. 規則 (5) は辺 (X,Y) の両端点が同じ色であってはならないことを表す.

$$1\{map(X,C) : color(C)\}1 \leftarrow vertex(X). \tag{4}$$
$$\leftarrow edge(X,Y), map(X,C), map(Y,C). \tag{5}$$

次に分解木において親子関係にあるバッグの局所解の合成について述べる. 本手法では D-FLAT [2,4] と同様に, 分解木に対する動的計画法を表すプログラムの実行を支援するため, プログラム実行中は, 与えられた分解木における現在処理中の頂点 x のバッグ (親バッグ) の局所解と, x のいずれかの子頂点のバッグ (子バッグ) の局所解とを実行エンジン (図 7) 内で保持している. 子バッグの局所解は一般に複数存在するので, $child\_row(R)$ によって, 「現在は子バッグの R 番目の局所解を処理中である」ことを表す.

局所解の合成は親バッグの局所解に対して子バッグの局所解のうち, 矛盾しないものを選択することで行う. 子バッグの局所解と親バッグの局所解とを区別をするために次のような述語が用意されていると仮定する (6.5 参照).

$child\_map(R,X,C)$: 子バッグの R 番目の局所解において頂点 X が色 C で塗られている

$accept(R)$ で, 子バッグの R 番目の解が親バッグの解と合成可能であることを表す. 対して $reject(R)$ で, R 番目の解が合成できないことを表す. これら 2 つの述語を定義する規則, すなわち, 解の合成を示す規則は以下の通りである:

$$reject(R) \leftarrow map(X,C), child\_map(R,X,C'), C != C'. \tag{6}$$
$$accept(R) \leftarrow not\ child\_reject(R), child\_row(R), not\ reject(R). \tag{7}$$

規則 (6) では親, 子バッグの局所解に共通して含まれる頂点について, 同じ色でないとき $reject(R)$ が生成される ($C != C'$ は C と C' が異なることを表す Clingo 記法). 合成可能となるのは任意の共通する頂点に関して同じ彩色が行われるときである. よって規則 (7) では合成が可能なことを表す $accept(R)$ が生成されるための必要条件として, 子バッグの R 番目の局所解が $reject(R)$ を持たないことを求めている.

3 色問題などの決定問題にはなく, コスト最大化問題に必要なコスト計算について述べる. コスト最大化問題の例として最大独立集合問題を再び取り上げる. ここでいうコストとは独立集合の頂点数を指す. 親, 子バッグの局所解を合成したときにコストの更新も必要である.

親バッグ, 子バッグの R 番目の各局所解の頂点数の和から共通する頂点数を減ずることでコストを更新する. まず子バッグの R 番目の解と親バッグの局所解に共通する頂点数を計算する規則を示す.

$$common(R,CC) \leftarrow child\_row(R), \quad CC = \#count\{X : in(X), child\_in(R,X)\}. \tag{8}$$

各局所解の構成要素 $in(X), child\_in(R,X)$(6.5 参照) の両方に含まれる頂点 X を数え上げる. その頂点数 CC に対して $common(R,CC)$ が生成される.

更新後のコストを計算する規則を以下に示す.

$$up\_cost(R,D) \leftarrow not\ reject(R), cost(C), child\_cost(R,C'), common(R,CC), D = C + C' - CC. \tag{9}$$
$$up\_cost(R,C) \leftarrow child\_row(R), reject(R), cost(C). \tag{10}$$

子バッグの R 番目の局所解が親バッグの局所解と合成可能な場合 (規則 (9)) は更新後のコスト D を各解のコスト C, C' の和か

ら共通部のコスト CC を減算した値と定義する. 合成ができない場合 (規則 (10)) は親バッグの解のコストは変更しない.

6.4 圧縮部分に対する問合せ

SLCFTG において非終端記号に関する生成規則の右辺に当たる部分グラフについて問合せを実行する. 各バッグでは圧縮されない部分グラフの問合せを行い, 圧縮部分の解と合成することで解を求める. 解の合成に関しては 6.3 と同様の手法で解集合プログラムを与える. 子バッグの局所解と同様, 圧縮部分に関する述語を接頭辞"$child$"を用いて表現する.

親, 子バッグの解の合成では, 合成可能な組み合わせを求めて Clingo 内で「一時的に」参照するだけであった. 対して圧縮部分とそれ以外の部分の解を合成するときは合成後の解を後の段階で一般に複数回使用する. そのため既存の解を更新しファクトとして実行エンジンに渡して「保存」しなければならない. 更新すべき情報は頂点 ("$vertex(X)$"), 解に関するファクト ($P(R, X_1, \ldots, X_n)$) である. これらの情報を表すファクト $new\_vertex(X)$, $new\_P(R, X_1, \ldots, X_n)$ を生成する規則を以下に示す.

$$new\_vertex(X) \leftarrow not\ vertex(X), child\_vertex(X). \quad (11)$$

$$new\_P(R, X_1, \ldots, X_n) \leftarrow not\ reject(R), not\ vertex(X), new\_P(R, X_1, \ldots, X_n). \quad (12)$$

規則 (11), (12) によって生成されるファクトは実行エンジンに渡され, 実行エンジンの内部状態が更新される.

圧縮部分とそれ以外の部分において共通する頂点は共通する解を導かれなければならない. そのために圧縮部分の解を合成する過程で既存の解が否定されることがある. 例えば 3 色問題では圧縮箇所で頂点 a, b について解として $map(a, red)$, $map(b, red)$ があるときそれ以外の箇所の解においても, これらのファクトを持つ解が存在しなければならない. 同一の解は少なくとも 1 つあればよい. 規則 (6), (7) によって解の合成可否を判定している. 非圧縮箇所の解に対応する, 圧縮箇所の矛盾しない解があるときは $accept(R)$ が生成される. すなわち $accept(R)$ が成り立たないときは解を削除する.

6.5 解集合プログラム中の組込み述語

表 6 実装システムが提供する組込み述語

名前	説明
$vertex(X)$	グラフに頂点 X が含まれる
$edge(X, Y)$	グラフに辺 (X, Y) が含まれる
$child\_vertex(X)$	子バッグに頂点 X が含まれる
$child\_reject(R)$	子バッグの R 番目の局所解が合成可能な解を持たない
$child\_row(R)$	子バッグの R 番目の局所解を処理中
$child\_P(R, X_1, \ldots, X_n)$	子バッグの R 番目の局所解において $P(X_1, \ldots, X_n)$ が成り立つ

分解木および, 圧縮データに対する問合せを処理するために実装システム側で提供する組込み述語を表 6 に示す. 実装システムは実行時に圧縮データを読み込み, "$vertex$", "$edge$" などグラフの情報を表す述語を用いたファクトを生成する. さらに, ユーザーが定義した述語 P に対して, 子バッグの R 番目の局所解において P が成り立つことを表す述語 $child\_P$ を自動的に定義してユーザーが利用可能とする機能を実現している.

6.6 評価実験

6.6.1 実装ツール・実験環境・ベンチマーク

5 節で使用したグラフデータのうち, huck および queen5 を使用した. これらのデータについての詳細は表 1, 表 2, 表 3 にそれぞれ示している. 更に頂点数 23, 辺数 71 である myciel4 を評価対象に加えている. myciel4 に対して木分解を適用した結果, 頂点数 12, 幅 12 の分解木が得られた. 圧縮前の分解木のファイルサイズは 1.41kB であり圧縮により, そのサイズは 0.44kB となった.

6.6.2 最大独立集合問題

myciel4 の最大独立集合問題の解 (頂点数) は 11 個, 他 2 つについては 5.4.3 参照. 表 7, 表 8 は各分解木についてそれぞれ最大独立集合の要素数算出を行ったときの実行時間, 最大使用メモリを示したものである. これらの表において直接問合せの欄の「入力処理」,「最大独立集合計算」の意味は表 4, 表 5 と同じである (5.4.3 参照).

表 7 問合せ時間

ファイル名	直接問合せ		解凍後問合せ	
	入力処理 (ms)	最大独立集合計算 (ms)	解凍 (ms)	最大独立集合計算 (ms)
myciel4	0.465	8094.55	3.043	7961.09
huck	1.29	1025.66	3.474	402.13
queen5	0.72	71560.1	6.654	16165.39

表 8 問合せ時の使用メモリ

ファイル名	直接問合せ		解凍後問合せ	
	入力処理 (kB)	最大独立集合計算 (kB)	解凍 (kB)	最大独立集合計算 (kB)
myciel4	5608	10268	4720	8584
huck	6872	9652	4752	7940
queen5	5696	11532	4754	8900

6.6.3 3 色問題

3 色問題の問合せについての結果を示す. 表 9, 表 10 は各分解木についてそれぞれ 3 色問題の問合せを行ったときの実行時間, 最大使用メモリを示したものである. 全データについて解は存在しない. すなわち 3 色の塗り方は存在しない.

表 9 問合せ時間

ファイル名	直接問合せ		解凍後問合せ	
	入力処理 (ms)	3 彩色計算 (ms)	解凍 (ms)	3 彩色計算 (ms)
myciel4	0.388	53.393	2.849	46.99
huck	1.057	363.98	3.496	42.41
queen5	0.754	1626.55	6.606	18.17

表 10　問合せ時の使用メモリ

ファイル名	直接問合せ		解凍後問合せ	
	入力処理 (kB)	3 彩色計算 (kB)	解凍 (kB)	3 彩色計算 (kB)
myciel4	5620	9536	4688	8068
huck	6680	9464	4680	6420
queen5	5708	10040	4708	7652

6.6.4　実行時間

最大独立集合問題, 3 色問題ともにデータの入力および解凍については共通の処理を行っている．解集合プログラムは問題によって異なるが, その読み込みにかかる時間は小さい．入力処理にかかる時間は直接問合せが解凍後問合せより小さい. 解凍にかかる時間によってこの差が生じる．

両問題とも問題の計算にかかる時間は解凍後問合せが直接問合せより小さい. 問合せ手法間で子バッグの局所解を親バッグの局所解と合成する処理に違いはない．一方, バッグの局所解を計算する処理は問合せ手法により異なる. 具体的には直接問合せでは圧縮箇所 (複数のバッグに共通する部分グラフ) について先に問合せ処理を実行する．次に非圧縮箇所の解を計算し, それらの解を合成することでバッグ全体の解を求めている．解の計算と合成は, まずデータに対して問合せプログラムとグラフデータの両方を Clingo の規則に変換する処理を行い, 次にソルバの Clingo を呼び出すことで実行している．最大独立集合問題の問合せを実行したときの Clingo の呼び出し回数を測定した結果を表 11 に示す．myciel4 についてはバッグ間に共通する部分グラフの圧縮がなされておらず, 問合せについても差異はない．他の 2 データは圧縮箇所に対して事前の問合せがなされているため, その局所解に対する合成処理に関しても Clingo が呼び出されている．データの変換と呼び出し, 実行にかかる時間が大きくなってしまうために, 全体の実行時間が増大してしまう．現在の実装では非圧縮部分の一つの解ごとに合成可能な解を求めている．これにより Clingo の呼び出し回数が増大してしまう. また圧縮効果の大きいデータにおいては, 圧縮箇所は多数存在する．圧縮箇所ごとに解の合成を実行しているため合成の処理回数が増大してしまっている．更に合成過程において削除される解が多いため, 最終的な解の個数に比べ Clingo の呼び出し回数が増大してしまう．

表 11　最大独立集合問題における Clingo 呼び出し回数

データ名	局所解の計算		解の合成
	直接問合せ	解凍後問合せ	
myciel4	12	12	2077
huck	396	34	294
queen5	6708	7	1272

6.6.5　最大使用メモリ量

最大独立集合問題, 3 色問題いずれも最大使用メモリ量は直接問合せにおいて大きい. 6.6.4 に示した実行時間と同様に局所解の合成処理は問合せ手法によって違いがなく, 使用メモリに影響はない．直接問合せについて局所解の計算時にその使用メモリ量が全プロセス中で最大となる．これは圧縮箇所の抽出によってグラフの情報が失われ, 後に削除される解が多く出現することが要因である．一方, 解凍後問合せについては解の合成時に使用メモリ量が最大となる．バッグごとの局所解を格納することによって使用メモリ量は単調に増大する．更に局所解の合成時に, 組み合わせ可能な解を記録しているために, 使用メモリ量が増大する．

7　あとがき

7.1　まとめ

本研究では, グラフに対する木分解により得られた分解木を効果的に圧縮可能な構造へ変換する手法を提案した．ここでは木文法を利用した圧縮を前提とし, 実装実験においては, 圧縮ツールとして TreeRePair を用いた. 提案した表現形式を用いることで効果的な圧縮が可能であることが示された．

更に圧縮データに対して解凍を行わず問合せを評価する 2 種の手法を提案し実装した. 問合せアルゴリズムを直接記述する手法と解集合プログラムの形式で与える手法である．

アルゴリズムを直接記述する手法においては最大独立集合問題を例にし評価実験を行った．実験の結果, 解凍せずに問合せを行う提案手法は, 解凍後問合せを行う手法に比べ実行時間において優れていることがわかった. 問合せアルゴリズムの記述に解集合プログラムを利用する手法においては最大独立集合問題に加え, 3 色問題を例に評価実験を行った. 今回の実装では直接問合せを行う場合より解凍後に問合せする場合が実行時間が小さくなった．

7.2　今後の課題

提案した分解木の表現形式は木の頂点数や幅が大きい木に対しては実行時間, データサイズが増大し変換が困難となる．これは分解木を提案する表現形式 (3 節) で出力する際に予め非終端記号への置換を一部実行することで, 重複部分の処理が低減できるため解消できると考えられる．

実社会で使用されるグラフデータは, それらに対する更新頻度が高い．更新を反映させるためには, 圧縮した木を解凍し, データを更新した後に再度圧縮する必要がある．木に対する圧縮法について解凍することなく直接更新を行う手法が提案されている [16]. これらの手法を基に圧縮された分解木を直接更新することで提案した圧縮法の実用性が高まることが期待できる．

アルゴリズムを問題ごとに記述した場合において直接問合せの実行時間は解凍後問合せに比べて実行時間は小さい．しかし改善が小さいグラフデータもあり, 効率に影響を与える要素を特定し, グラフの構造に応じた手法の改善が見込めると考えられる．

アルゴリズムの記述に解集合プログラムを利用した場合において直接問合せの実行時間が大きい．その要因は圧縮箇所を利用した解の合成にかかる Clingo の呼び出し回数の増大が原因であった．現在の実装は個別の解ごとに Clingo が呼び出されているが, 全ての解を同時に処理することで実行時間を短縮できると考えられる．

謝辞

木圧縮に関して種々ご助言を頂いた名古屋大学助教 橋本健二氏, ならびに, 木分解ツール htd および解集合プログラムソルバ Clingo についてご教示頂いた名古屋大学教授 番原睦則先生に感謝いたします.

参考文献

[1] M. Abseher. https://github.com/mabseher/htd
[2] M. Abseher, B. Bliem, G. Charwat, F. Dusberger, M. Hecher and S. Woltran, The D-FLAT System for Dynamic Programming on Tree Decompositions, JELIA 2014, LNAI 8761, 558–572.
[3] M. Besta and T. Hoefler, Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations, CoRR abs/1806.01799, 2018.
[4] B. Bleim, M. Morak and S. Woltran, D-FLAT: Declarative Problem Solving Using Tree Decompositions and Answer-Set Programming, Theory Prac. Log. Program, 12(4-5), 445–464, 2012.
[5] H. L. Bodlaender, TreewidthLIB. http://www.cs.uu.nl/research/projects/treewidthlib
[6] P. Boldi and S. Vigna, The Webgraph Framework I: Compression Techniques, 13th International Conference on World Wide Web, 595–602, 2004.
[7] N. R. Brisaboa, S. Ladra and G. Navarro, K^2-trees for Compact Web Graph Representation, 16th International Symposium on String Processing and Information Retrieval, 18–30. 2009.
[8] G. Buehrer and K. Chellapilla, A Scalable Pattern Mining Approach to Web Graph Compression with Communities, International Conference on Web Search and Data Mining, 95–106, 2008.
[9] F. Claude and S. Ladra, Practical Representations for Web and Social Graphs, 20th ACM Conference on Information and Knowledge Management, 1185–1190, 2011.
[10] H. Comon, et al., Tree Automata Techniques and Applications, 2008. http://tata.gforge.inria.fr/
[11] R. Downey and M. R. Fellows, Fundamentals of Parameterized Complexity, Chapter 10 Tree Width and Dynamic Programming, 2013.
[12] W. Fan, J. Li, X. Wang and Y. Wu, Query Preserving Graph Compression, ACM Special Interest Group on Management of Data, 157–168, 2012.
[13] J. Fischer and D. Peters, GLOUDS: Representing Tree-Like Graphs, Journal of Discrete Algorithms, 36, 39–49, 2016.
[14] A. P. Francisco, T. Gagie, S. Ladra and G. Navarro, Exploiting Computation-Friendly Graph Compression Methods for Adjacency-Matrix Multiplication, Data Compression Conference, 307–314, 2018.
[15] M. Gebser, R. Kanminski, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, S. Thiele and P. Wanko, Potassco guide version 2.2.0.
[16] K. Hashimoto, R. Takayama and H. Seki, Direct Update of XML Documents with Data Values Compressed by Tree Grammars, IEICE Transactions on Information and Systems, E101-D(6), 1467–1478, June 2018.
[17] C. Hernández and G. Navarro, Compressed Representations for Web and Social Graphs, Knowledge and Information Systems, 40, 279–313, 2014.
[18] M. Jark, R. Pichler and S. Wotran, Answer-Set Programming with Bounded Treewidth, IJCAI 2009, 816–822.
[19] U. Kang, C. E. Tsourakakis and C. Faloutsos, PEGASUS: Mining Peta-Scale Graphs, Knowledge and Information Systems, 303–325, 2011.
[20] M. Lohrey, S. Maneth and R. Mennicke, XML Tree Structure Compression Using RePair, Information Systems, 38, 1150–1167, 2013.
[21] S. Maneth and F. Peternek, A Survey on Methods and Systems for Graph Compression, CoRR abs/1504.00616, 2015.
[22] S. Maneth and F. Peternek, Compressing Graphs by Grammars, IEEE 32nd International Conference on Data Engineering, 109–120, 2016.
[23] F. Merz and P. Sanders, PReaCH: A Fast Lightweight Reachability Index Using Pruning and Contraction Hierarchies, European Symposium on Algorithms, 701–712, 2014.
[24] H. Yildirim, V. Chaoji and M. J. Zaki, GRAIL: A Scalable Index for Reachability Queryies in Very Large Graphs, The VLDB Journal, 21, 509–534, 2012.
[25] 小島和之，関浩之, 木分解と木文法圧縮を利用したグラフ圧縮法および圧縮データへの直接問合せ処理法の提案と評価，電子情報通信学会データ工学研究会，DE2020-13, 2020.09.
[26] 小島和之, 関浩之, 木分解の圧縮および解集合プログラミングによる問合せ, 電子情報通信学会データ工学研究会, DE2021-16, 2021.12.