DEIM Forum 2021 A13-4

連言経路検索のための構造索引技術

佐々木勇和† George Fletcher†† 鬼塚 真†

† 大阪大学大学院情報科学研究科
†† Eindhoven University of Technology
E-mail: †{sasaki,onizuka}@ist.osaka-u.ac.jp, ††g.h.l.fletcher@tue.nl

あらまし 連言経路検索 (Conjunctive path queries, *CPQ*) は，グラフ分析で頻繁に用いられているグラフ問合せ言語である．*CPQ* の高速化に有効な索引技術は存在せず，問合せ時間の削減が課題である．本論文では，*CPQ* に特化させた構造索引技術 CPQx を提案する．CPQx は，*path-bisimulation* に基づき索引のブロックを分割する．*path-bisimulation* により分割した経路は，*CPQ* では区別できないため，効率的な枝刈りが可能である．CPQx に関して，索引の構築，索引のメンテナンス，および索引を用いたクエリ処理アルゴリズムを開発する．さらに，ユーザが興味のあるナビゲーションパターンに基づく *interest-aware* CPQx を提案し，索引構築のコストを低減しつつ，さらなるクエリ処理の高速化を実現する．実データを用いた実験を行い，既存技術と比較して，より小さい索引サイズで最大数千倍の高速化を達成することを示す．また，ソースコードを公開し，さらなる研究の発展に寄与する．

キーワード グラフデータベース, サブグラフマッチング, bisimulation

1 はじめに

グラフはデータオブジェクトとそれらの関係性を分析する様々な応用に用いられている．例えば，知識グラフや，ソーシャルネットワーク，化学データベースは典型的な応用例である [1]．多くのグラフでは，節点間の関係性を表すために，枝にラベルが付与されている．例えば，ソーシャルネットワークでは，ユーザやブログが節点となり，ユーザ間の枝は友達関係を表すラベルおよびユーザとブログ間の枝は作成やブックマークを表すラベルが付与される．

グラフ経路の分析は，基礎的な分析のひとつである．連言経路問合せ (Conjunctive path query, CPQ) は，多くの問合せ言語にてサポートされている基本的なグラフ問合せであり，経路ナビゲーション，閉路，およびそれらの連言から構成されている [1]．問合せログ分析 [2] では，CPQ は実用上の 99%以上の問合せにおけるグラフ形状を表現できることが知られている．

グラフデータの大規模化に伴い，CPQ を効率的に処理可能な索引技術が求められている．構造索引 [1] は，CPQ の高速化に用いることができ，与えられたラベル系列に対応した経路をマテリアライズする．

構造索引を問合せ言語に特化しているかという観点から再整理する．問合せ言語に特化している構造索引では，言語とデータモデル特有の構造を用いて索引を設計し，特定の経路問合せを高速化する．例えば，DataGuides [3] は，根付き半構造グラフデータにおける正規経路問合せ *RPQ* に特化している．一方で，問合せ言語に特化していない構造索引は，単純な経路ナビゲーションの検索のために設計されており，幅広い経路問合せに利用できるが，複雑な経路問合せにおいて十分な高速化できない．

問合せ言語に特化した構造索引は，経路の集合を等価クラスに分割し，等価クラスに基づいて索引が構築される．等価クラスとは特定の問合せ言語では分割できない経路の集合であり，問合せ結果に含まれない経路の枝刈りに有効である．問合せ言語に特化した構造索引は多く提案されているが，CPQ に特化した構造索引は存在しない．

CPQ の等価クラスの分割として，path-bisimilarity が知られている [4]．path-bisimilarity は，CPQ の表現能力と密接に関連しており，等価な分割を満たすことができる．しかし，path-bisimilarity を CPQ 処理のための構造索引設計への実践的な活用に関する研究は存在しない．

研究課題． 既存の特定言語に特化した索引構造技術が CPQ に適応不可能であり，path-bisimulation に基づく索引構造が実グラフに対して実際に有効であるかは非自明である．CPQ の構造的な特徴を活かした索引技術は，CPQ を劇的に高速化できる可能性がある一方で，いくつかの解決すべき課題がある．まず，CPQ の形式的な表現能力の特徴を索引技術と問合せ処理アルゴリズムに実践的に組込む必要があるが，索引の設計，構築，および利用は非自明な課題である．さらに，path-bisimulation に基づく分割が効率的に計算可能か，コンパクトに表現可能か，メンテナンスができるのかを理論的に求める必要がある．path-bisimulation を効率的に計算可能な技術は存在せず，既存技術は節点に基づく bisimulation 技術しか存在しない [5].

貢献． 本研究では，CPQ における構造索引技術に関する初めての研究として，7 つの貢献を実現する．まず，(1) CPQ に特化した構造索引 CPQx を提案する．CPQx は，path-bisimulation [4] に基づくため，CPQ の表現能力と密接に関連している．CPQx の利用をサポートするために，(2) 索引の効率的な構築，(3) 索引のメンテナンス，および (4) 索引を利用した問合せアルゴリズムを開発する．これらの貢献は，シンプルかつ効果的な索引の設計と検索結果の正当性を保証により実現される．さら

に，多くの現実の応用では，ユーザの経路に関する興味が既に決まってる場合がある．つまり，ユーザは事前に特定の経路ナビゲーションに関して問合せすることが決まっていることがある．そこで，(5) 興味に基づく索引構造技術 iaCPQx を開発し，索引構築時間とメモリ使用量の削減と問合せ処理のさらなる高速化を実現する．これらの開発した技術を，(6) 多様な実グラフを用いた実験により有効性を検証する．既存研究と比較して，最大で数千倍の問合せ処理の高速化をできることを示す．最後に，(7) C++のソースコードを公開することにより，さらなる研究の発展に寄与する [1]．

2 関 連 研 究

経路問合せのための構造索引技術は XML や半構造データにおいて頻繁に研究されている [6]．しかし，既存の言語に特化した構造索引技術は，CPQ や一般的なグラフデータに焦点を当てていない．例えば，DataGuides [3], A[k]-index [7], および T-index [6] は，根付き半構造グラフデータにおける正規経路問合せ *RPQ*，P(k)-index [8] は，木構造データにおける XPath 問合せに焦点を当てている．これらの構造索引技術は二つの以下の二つの理由により，一般的なグラフにおける CPQ には適応できない．まず，既存技術は，根付き半構造グラフデータと木構造データを対象にしているため，これらの索引構築技術は一般的なグラフに適応できない．次に，RPQ と XPath 問合せは，閉路や経路の連言をサポートしておらず，これらの問合せの特徴は CPQ に適応できない．これらの理由により，既存技術は，閉路を持つような任意のグラフにおいて，閉路の連言に対して対応することができない．

準同型サブグラフマッチングや RDF エンジンは CPQ 問合せに用いることができる．そこで，効率的な手法として知られている Tentris [9] および TurboHom++ [10] を評価実験において比較手法として用いる．

3 事 前 知 識

本章では，グラフと CPQ の定義について述べ，その後，構造索引技術 [11] について述べる．

3.1 グラフ，経路，ラベル系列

グラフ $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ は，有限の節点集合 $\mathcal{V}$，ラベル付きの枝集合 $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{L}$ から構成され，$(v, u, \ell) \in \mathcal{E}$ は，ラベル $\ell \in \mathcal{L}$ をもつ v から u への枝を表す．逆方向の枝も処理対象とするために，$\mathcal{E}$ with ℓ^{-1} $\ell \in \mathcal{L}$ に対して ℓ^{-1} と $(v, u, \ell) \in \mathcal{E}$ に対して (u, v, ℓ^{-1}) が，それぞれ $\mathcal{L}$ および $\mathcal{E}$ に含まれているとする．

節点のペア $(v, u) \in \mathcal{V} \times \mathcal{V}$ を経路とし，v と u をそれぞれ経路の始点と終点とする．$\mathcal{P}^{\leqq k}$ $(k \geqq 0)$ を始点から終点までの経路長が k 以下となる経路集合とする．ここで，$\mathcal{P}^{\leqq k} \subseteq \mathcal{V} \times \mathcal{V}$ であることに注意されたい．

1：https://github.com/yuya-s/CPQ-aware-index

非負の k に対して，k 個のラベルの系列を長さ k のラベル系列とする．長さ k 以下のラベル系列の集合を $\mathcal{L}^{\leqq k}$ とし，ラベル系列を $\bar{\ell} = \langle \ell_1, \ldots, \ell_j \rangle$ $(j \leqq k)$ とする．さらに，$\mathcal{L}^{\leqq k}(v, u)$ を v から u 上の長さ k 以下の経路上のラベル系列の集合とする．γ を全ての $(v, u) \in \mathcal{P}^{\leqq k}$ に対する $\mathcal{L}^{\leqq k}(v, u)$ の平均サイズとする．

3.2 連言経路問合せ

連言経路問合せ CPQ は，ラベル 'ℓ'，結合 '$\circ$'，連言 '$\cap$'，および同一 'id'，の操作の繰り返しにより表現され，下記の文法で表記される．

$$CPQ ::= id \mid \ell \mid CPQ \circ CPQ \mid CPQ \cap CPQ \mid (CPQ).$$

$q \in CPQ$ とし，$\mathcal{G}$ における q の問合せ結果を $[\![q]\!]_{\mathcal{G}}$ としたとき， それぞれの操作を下記の式で表す．

$$\begin{aligned}
[\![id]\!]_{\mathcal{G}} &= \{(v, v) \mid v \in \mathcal{V}\},\\
[\![\ell]\!]_{\mathcal{G}} &= \{(v, u) \mid (v, u, \ell) \in \mathcal{E}\},\\
[\![q_1 \circ q_2]\!]_{\mathcal{G}} &= \{(v, u) \mid \exists m \in \mathcal{V} : (v, m) \in [\![q_1]\!]_{\mathcal{G}} \\
&\qquad \text{and } (m, u) \in [\![q_2]\!]_{\mathcal{G}}\},\\
[\![q_1 \cap q_2]\!]_{\mathcal{G}} &= \{(v, u) \mid (v, u) \in [\![q_1]\!]_{\mathcal{G}} \text{ and } (v, u) \in [\![q_2]\!]_{\mathcal{G}}\},\\
[\![(q_1)]\!]_{\mathcal{G}} &= [\![q_1]\!]_{\mathcal{G}}.
\end{aligned}$$

問合せ $q \in CPQ$ に対して，q の**直径** $\mathrm{dia}(q)$ は次のように計算する．問合せの直径は，結合されるラベルの最大数である．同一操作は直径 0; ラベルは直径 1; $\mathrm{dia}(q_1 \cap q_2) = \max(\mathrm{dia}(q_1), \mathrm{dia}(q_2))$; および，$\mathrm{dia}(q_1 \circ q_2) = \mathrm{dia}(q_1) + \mathrm{dia}(q_2)$.

3.3 特定の言語に特化していない経路索引

言語に特化した索引は，問合せ言語 L とデータベースインスタンス I が与えられたときに，I のデータオブジェクトを L 等価，つまり，分割されたデータオブジェクトは L では分割することはできない（L のいかなる問合せにおいて，それらのデータオブジェクトは全て結果に含まれるか，全て含まれないかのどちらかとなる）．

特定の言語に特化していない経路索引 [11] はラベル系列を検索キーとして経路集合を出力する転置索引である．この索引の欠点として，経路が複数回索引内に格納されることにより索引サイズの増加と閉路と連言に対する非効率性がある．経路索引のサイズは，それぞれの経路は平均で γ 回格納されるため，$O(\gamma|\mathcal{P}^{\leqq k}|)$ となる．

4 構造索引 CPQx

本章では，構造索引 CPQx，CPQx を用いた効率的な問合せ処理アルゴリズム，およびグラフ更新に対する CPQx のメンテナンス方法について述べる．

4.1 アイデアの概要

CPQx の基本的な考えは，与えられたグラフ $\mathcal{G}$ における経路集合 $\mathcal{P}^{\leqq k}$ を CPQ_k では分割不可能かつ非重複なブロックに

分割することである．経路の分割は，CPQ_k の表現能力に密接な *k-path-bisimulation* (Theorem 41 参照) に基づいて行う．もし二つの経路が k-path-bisimilar であるならば，いかなる $q \in CPQ_k$ において二つの経路を区別できないという性質をもつ．つまり，二つの経路の両方が問合せ結果に入るか，どちらも入らないかのいずれかになる．CPQx の構築において，多項式時間の時間と計算複雑性により実現可能な k-path-bisimulation に基づく分割の効率的なアルゴリズムを開発する (4.3 章)．

構造索引 CPQx は，二つのデータ構造 I_{l2h} and I_{h2p} から構成される (4.2 章)．I_{l2h} は，ラベル系列からブロックの識別子集合へのマッピング関数，および I_{h2p} はブロックの識別子から経路集合へのマッピング関数である．CPQ 問合せ処理では二つのデータ構造を用いて 2 段階の検索により行う (4.4 章)．まず，第一段階では，ラベル系列からブロック集合を検索し，問合せに無関係な経路集合を枝刈りすることができる．次に，第二段階では，第一段階得たブロックに対して，ブロックに対応する経路集合を検索する．この問合せ処理アルゴリズムにおける利点は，ブロック識別子の連言をとることが可能であり，経路に対する連言無しで CPQ の連言操作を実現することができる点である．ブロック識別子の数は，経路数に比べて格段に少ないため，問合せ処理の高速化が可能である．

4.2 構造索引 CPQx の定義

CPQx は，*k-path-bisimulation* による経路の等価性に基づいて構築される．k-path-bisimulation は以下の定義で表すことができる．

［Definition 41］（k-path-bisimulation） グラフ $\mathcal{G}$，非負の整数 k，および $v, u, x, y \in \mathcal{V}$ が与えられたとき，下記の条件を満たすとき，経路 (v, u) と (x, y) は *k-path-bisimilar* $(v, u) \approx_k (x, y)$ である．

（1） $v = u$ であるときに限り $x = y$ であり;

（2） もし $k > 0$ の場合，それぞれの $\ell \in \mathcal{L}$ に対して，

- （a） もし $(v, u, \ell) \in \mathcal{E}$ の場合，$(x, y, \ell) \in \mathcal{E}$; かつ，もし $(u, v, \ell) \in \mathcal{E}$ の場合，$(y, x, \ell) \in \mathcal{E}$ である;
- （b） もし $(x, y, \ell) \in \mathcal{E}$ の場合，$(v, u, \ell) \in \mathcal{E}$; かつ，もし $(y, x, \ell) \in \mathcal{E}$ の場合，$(u, v, \ell) \in \mathcal{E}$ であり，

（3） もし $k > 1$ の場合，

- （a） それぞれの $m \in \mathcal{V}$ に対して，もし (v, m) と (m, u) が $\mathcal{P}^{\leq k-1}$ に含まれる場合， (x, m') と (m', y) が $\mathcal{P}^{\leq k-1}$ に含まれるような $m' \in \mathcal{V}$ が存在し，さらに，$(v, m) \approx_{k-1} (x, m')$ と $(m, u) \approx_{k-1} (m', y)$ であり;
- （b） それぞれの $m \in \mathcal{V}$ に対して，もし (x, m) と (m, y) が $\mathcal{P}^{\leq k-1}$ 含まれる場合， (v, m') と (m', u) が $\mathcal{P}^{\leq k-1}$ に含まれるような $m' \in \mathcal{V}$ が存在し，さらに，$(x, m) \approx_{k-1} (v, m')$ と $(m, y) \approx_{k-1} (m', u)$ である．

直観的には，経路 (v, u) と (x, y) は，v から u および x から y における長さ k 以下のいかなる経路において全て等しいラベル系列かつ閉路パターンがある場合に，k-path-bisimilar である．

k-path-bisimulation は，以下の定理より CPQ_k の構造的な表現能力とすることができる [4].

［Theorem 41］ グラフ $\mathcal{G}$，非負の整数 k，および $v, u, x, y \in \mathcal{V}$ が与えられたとき，もし $(v, u) \approx_k (x, y)$ であるならば，全ての $q \in CPQ_k$ に対して，$(x, y) \in [\![q]\!]_{\mathcal{G}}$ の場合のみ $(v, u) \in [\![q]\!]_{\mathcal{G}}$ となる．

定理 41 を構造索引設計に活用するために，k-path-bisimulation に基づく経路の等価クラスを定義する．$\mathcal{P}^{\leq k}$ の等価クラスへの分割は，索引におけるブロックとなる．

［Definition 42］ グラフ $\mathcal{G}$，$v, u \in \mathcal{V}$，非負の整数 i，および，$(v, u) \in \mathcal{P}^{\leq i}$ が与えられたとき，(v, u) の i-path-bisimulation 等価クラスは以下とする．

$$[(v, u)]_i(\mathcal{G}) = \{(x, y) \mid x, y \in \mathcal{V} \text{ and } (v, u) \approx_i (x, y)\}.$$

等価クラスをブロックと呼び，ブロックの集合を以下とする．$B_i(\mathcal{G}) = \{[(v, u)]_i(\mathcal{G}) \mid v, u \in \mathcal{V}\}$.

定理 41 の補題として，問合せ処理は $B_k(\mathcal{G})$ と密接に関連していることを示す．

［Corollary 41］ グラフ $\mathcal{G}$，非負の整数 k，および $q \in CPQ_k$ が与えられたとき，$[\![q]\!]_{\mathcal{G}} = \bigcup_{block \in B} block$ となる，$B \subseteq B_k(\mathcal{G})$ が存在する．

補題 41 を問合せ処理に活用するために，識別子 $b_i(v, u)$ を B_i の各ブロック $[(v, u)]_i$ に割り当てる．二つの経路が同じブロックに入る場合，それらの経路は同じブロック識別子をもつ．経路が i-path-bisimilar であるならば，その経路は $(i-1)$-path-bisimilar と必ずなるため，もし経路が B_i の同じブロックに入ってる場合，B_{i-1} においても同じブロックに入ってる．

k-path-bisimulation 分割を下記に定義する．

［Definition 43］ 非負の整数 k に対して，グラフ $\mathcal{G}$ の k-path bisimulation 分割は下記とする．

$$[\mathcal{G}]_k = \{B_i(\mathcal{G}) \mid 0 \leq i \leq k\}.$$

それぞれの経路 $(v, u) \in \mathcal{P}^{\leq k}$ は，k 個のブロック識別子からなる系列 $\langle b_0(v, u), b_1(v, u), \ldots, b_k(v, u)\rangle$ が割り当てられる．このブロック識別子の系列を *history* と呼ぶ．k-path-bisimilar 経路は，同じ history をもつことは自明である．それぞれの history は，他と異なるブロック識別子の系列をもつため，各 history に history 識別子を割り当て，$\mathcal{H}$ を history 識別子の集合と定義する．さらに， $\mathbb{P}(h) \subseteq \mathcal{P}^{\leq k}$ を h に属する経路集合，$\mathcal{H}(\bar{\ell}) \subseteq \mathcal{H}$ を $\bar{\ell}$ に属する経路集合とする．

k-path-bisimulation 分割に基づいて，構造索引 CPQx を下記と定義する．

［Definition 44］（構造索引 CPQx） $[\mathcal{G}]_k$ が与えられたとき，$\text{CPQx}_{[\mathcal{G}]_k}$ は，ラベル系列から history 識別子の集合へのマップを表すデータ構造 I_{l2h} と history 識別子から経路へのマップを表すデータ構造 I_{h2p} のペアとする．

$$I_{l2h}(\bar{\ell}) = \{h \mid h \in \mathcal{H}(\bar{\ell})\},$$
$$I_{h2p}(h) = \{(v, u) \mid (v, u) \in \mathbb{P}(h)\}.$$

CPQx は，構造索引 [11] より小さいサイズとなる．これは，提案する構造索引では，それぞれの経路が一つの history と関連するのに対して，経路索引はそれぞれの経路が複数のラベル系列に関連するためである．そのため，我々の構造索引は，問合せ結果となる経路集合をより少ないアクセスで効率的に求めることができる．

[Theorem 42] CPQx のサイズは $O(\gamma|\mathcal{H}| + |\mathcal{P}^{\leqq k}|)$ となる．

証明: I_{l2h} はそれぞれのラベル系列に対応して history 識別子の集合を格納している．ラベル系列に対して，history 識別子は平均で γ 個格納されている．そのため，I_{l2h} のサイズは $O(\gamma|\mathcal{H}|)$ となる．I_{h2p} においては，それぞれの経路は重複せずに一度しか格納されていないため，I_{h2p} のサイズは $O(|\mathcal{P}^{\leqq k}|)$ となる．そのため，CPQx のサイズは $O(\gamma|\mathcal{H}| + |\mathcal{P}^{\leqq k}|)$ となる． □

$|\mathcal{H}|$ は最大で $|\mathcal{P}^{\leqq k}|$ あるため，CPQx のサイズ $O(\gamma|\mathcal{H}| + |\mathcal{P}^{\leqq k}|)$ は，構造索引 [11] のサイズ $O(\gamma|\mathcal{P}^{\leqq k}|)$ より小さい．

4.3 索 引 構 築

本節では，効率的な索引構築方法について述べる．ここでの主たる貢献は，時間と空間複雑性がともに多項式時間となる k-path-bisimulation 分割の計算方法の開発である．k-path bisimulation 分割を計算後に，$I_{[\mathcal{G}]_k} = (I_{l2h}, I_{h2p})$ を構築する．

k-path-bisimulation 分割の計算アルゴリズムは，ボトムアッププアプローチを用いる．これは，i-path-bisimulation が $(i-1)$-path-bisimulation から求められるためである．二つの経路は下記の条件を満たす場合，k-path-bisimilar である; (1) B_{i-1} と B_1 において二つの経路が同じブロック識別子をもつ，および (2) 両方の経路が閉路であるもしくは両方とも閉路ではない．さらに，$1 \leqq i \leqq k$ の i-path-bisimulation を効率的に計算するために， 長さ i のラベル系列で接続していない経路の等価クラスの計算をスキップする．このような経路の数は膨大であるため，計算時間の大幅な短縮が可能である．経路の history $\langle b_0(v,u), b_1(v,u), \ldots, b_k(v,u)\rangle$ は，もし部分的にもしくは全てのブロック識別子が経路に割り当てられていなくても求めることができる．ボトムアッププアプローチは，k を変えることにより複雑性を調整できるため，k を計算機資源に基づいて決めることも可能である．history から history 識別子を生成し，$\bar{\ell}$ and $h \in \mathcal{H}(\bar{\ell})$ のペアを I_{l2h} に挿入，history 識別子 h と $(v,u) \in \mathbb{P}(h)$ のペアを I_{h2p} 挿入する．

Algorithms 1 と 2 は k-path-bisimulation 分割の計算と索引構築の疑似コードをそれぞれ表す．Algorithm 1 では，効率的なブロック識別子の割り当てのために，i-path-bisimilar 経路がシーケンシャルに並ぶように $\mathbb{S}^i$ の要素をソートする．Algorithm 2 では，(v,u) の history 識別子 h をハッシュ関数を用いて割り当てる．もし二つの経路が同じ history をもつなら，同じ history 識別を割り当てる．

索引構築の時間複雑性と空間複雑性について以下の定理が満たされる．

[Theorem 43]（空間複雑性） グラフ $\mathcal{G}$ と正の整数 k が与えられたとき，索引構築の空間複雑性は $O((k+d)|\mathcal{P}^{\leqq k}| + \gamma|\mathcal{H}|)$ となる．ここで，d は最大の次数を表す．

Algorithm 1: Computing k-path-bisimulation

```
input : Graph G, natural number k
output: [G]_k
procedure kPathBisimulation(G, k)
S^i_(v,u) = ∅ for i = 1,...,k and
  ∀(v,u) ∈ P^≦i − P^≦i−1 + (P^≦i ∩ P^≦i−1);
for e = (v,u,ℓ) ∈ E do
    S^1_(v,u) ← S^1_(v,u) ∪ {ℓ};
Sort S^1 according to edge labels and (v,u);
Set b_1(v,u) for ∀(v,u) ∈ P^≦1 as B_1;
for i = 2,...,k do
    for ∀S^i−1_(v,m) do
        for ∀S^1_(m,u) do
            S^i_(v,u) ← S^i_(v,u) ∪ {b_i−1(v,m), b_1(m,u)};
    Sort S^i according to block identifiers and (v,u);
    Set b_i(v,u) for
      ∀(v,u) ∈ P^≦i − P^≦i−1 + (P^≦i ∩ P^≦i−1) as B_i;
    if i ≠ 2 then Clear S^i−1:
return [G]_k = {B_1,...,B_k};
end procedure
```

Algorithm 2: Construction of the structural index

```
input : Graph G, natural number k
output: Structural index I_[G]_k = {I_h2p, I_l2h}
procedure StructuralIndex(G, k)
[G]_k ← kPathBisimulation(G, k);
for (v,u) ∈ P^≦k do
    h ← hash(⟨b^1_v,u,...,b^k_v,u⟩);
    if h is NULL then
        h ← h_new;
        hash(⟨b^1_v,u,...,b^k_v,u⟩) ← h ;
        H ← H ∪ {h};
        Update h_new;
    I_h2p.append(h,(v,u));
for h ∈ H do
    for (v,u) ∈ I_h2p(h) do
        for ℓ̄ ∈ L^≦k(v,u) do
            I_l2h.append(ℓ̄,h);
sort (v,u) in I_h2p and h in I_l2h;
return I_[G]_k;
end procedure
```

[Theorem 44]（時間複雑性） グラフ G と正の整数 k が与えられたとき，索引構築の時間複雑性は $O(k(d|\mathcal{P}^{\leqq k}| + |\mathcal{P}^{\leqq k}|\log|\mathcal{P}^{\leqq k}|) + \gamma|\mathcal{H}|\log\gamma|\mathcal{H}|)$ となる．ここで，d は最大の次数を表す．

4.4 問合せ処理アルゴリズム

構造索引を用いて問合せ処理の高速化を実現する．このとき，構造索引のみを用いて，元のグラフは一切必要としない．問合せ処理は，効果的に history を用いて，検索結果に入らない経

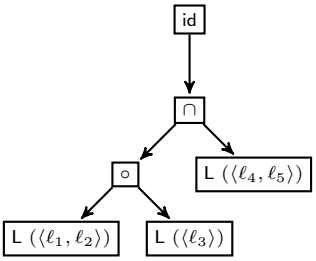


図 1: $k = 2$ における，問合せ $[(\ell_1 \circ \ell_2 \circ \ell_3) \cap (\ell_4 \circ \ell_5)] \cap id$ のパース木.

路に対するコストを削減する.

問合せ処理では，パース木に従って問合せを処理する．このとき，ラベル系列は，最大で k サイズとなるように分割される (図 1 参照)．パース木のそれぞれのノードは，論理的な操作を表し，操作は LookUp, Conjunction, Join, and Identity の 4 つである．パース木の根ノードから処理を開始，左から右に処理する．本稿では，実行計画はヒューリスティックに導出しており，問合せの最適化は今後の課題である.

問合せ処理アルゴリズムは， path-bisimulation 分割を有効活用することで，特に Conjunction と Identity を高速化できる．同じ history 識別子をもつ経路は，k-path-bisimilar 経路であるため，ある history 識別子が二つのラベル系列から導出される history 識別子の集合に両方とも含まれるなら，その history 識別子から導出される経路集合は両方のラベル系列を満たす経路である．つまり，経路集合の比較無しで，経路の連言を求めることができる．Identity においては，k-path-bisimilar 経路は閉路か閉路ではないかに基づくため，history 識別子から導出される経路集合のうち，ある一つの経路を確認するだけで，その経路集合が全て閉路か全て閉路ではないかを求めることができる．つまり，全ての経路を確認する必要がないため，高速化が可能である．history 集合のサイズは経路集合のサイズに比べて格段に小さいため，劇的な高速化が可能である.

[Proposition 41]（連言の正しさ） 二つの history 集合 $\mathbb{H}$ と $\mathbb{H}'$ が与えられてとき，$h \in \mathbb{H} \cap \mathbb{H}'$ に対応する経路集合 $\mathbb{P}(h)$ は，全ての $h \in \mathbb{H}$ と $h' \in \mathbb{H}'$ に対応する $\mathbb{P}(h) \cap \mathbb{P}(h')$ と同じである.

5 興味に基づく構造索引 iaCPQx

多くの応用において，ユーザは事前に特定のラベル系列に着目した分析を目的としていることがある．そのため，CPQx ではラベルの逆も含めて長さ k 以下の全てのラベル系列を保持するが，意図する分析に必要としないことがありうる．この動機に基づき，興味に基づく構造索引 iaCPQx を開発する．この索引では，全ての CPQ を処理可能であるが，興味ラベル系列を含む場合に大きく高速化することができる.

索引設計. 興味に基づく構造索引において，まず *interest-aware path-equivalence* を下記のように定義する.

[Definition 51]（Interest-aware Path-Equivalence） グラフ $\mathcal{G}$，節点 $v, u, x, y \in \mathcal{V}$，およびラベル系列の集合 $\mathcal{L}_q \subseteq \mathcal{L}^{\leq k}$ が与えられたとき，経路 (v, u) と (x, y) は下記の条件を満た場合，*interest-aware path-equivalent* $(v, u) \approx_i (x, y)$ とする.

（1） $v = u$ の場合に限り $x = y$ であり;

（2） $\mathcal{L}^{\leq k}(v, u) \cap \mathcal{L}_q = \mathcal{L}^{\leq k}(x, y) \cap \mathcal{L}_q$.

$\mathcal{L}_q$ は，特定のラベル系列の集合を表す．このとき，$\mathcal{L}_q$ は長さ 1 の全てのラベル系列（つまり，全ての枝のラベル）を含むことにより，いかなる問合せに答えることができる.

CPQx と iaCPQx の違いは，まず前者が同じ history 識別子は k-path bisimilar 経路の集合に割り当てられ，後者は interest-aware path-equivalent 経路に割り当てれる点である．interest-aware path-equivalence は，k-path bisimulation より弱い定義であるため，より多くの経路が同じ history 識別子をもつこととなる．これにより，iaCPQx のサイズはより小さくなり，高速な問合せ処理が可能となる．iaCPQx のクエリ処理やメンテナンスは興味ラベル集合の取り扱いを含む以外は CPQx と同様の手順となる.

表 1: データセットの統計：$|\mathcal{E}|$ と $|\mathcal{L}|$ はそれぞれ枝とラベルの逆方向も含む.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{L}\rvert$	Real label?
Robots	1,484	5,920	8	✓
ego-Facebook	4,039	176,468	16	
Advogato	5,417	1,724,554	8	✓
Youtube	15,088	21,452,214	10	✓
StringHS	16,956	2,483,530	14	✓
StringFC	15,515	4,089,600	14	✓
BioGrid	64,332	862,277	14	✓
Epnions	131,828	1,681,598	16	
WebGoogle	875,713	10,210,074	16	
WikiTalk	2,394,385	10,042,820	16	
YAGO	4,295,825	24,861,400	74	✓
CitPatents	3,774,768	33,037,896	16	
Wikidata	9,292,714	110,851,582	1054	✓
Freebase	14,420,276	213,225,620	1556	✓

6 評価実験

提案技術の実験結果について述べる．実験には，512GB メモリと Intel(R) Xeon(R) CPU E5-2699v3 @ 2.30GHz processor を搭載する Linux 計算機を用い，全てのアルゴリズムはシングルスレッドで動作させる.

データセット. 表 1 は，本実験で用いたデータセットの概観を示す．データセットは，実ラベルを伴う 9 つのデータとラベルを伴わない 5 つのデータを含む．グラフは，化学データベースや，ソーシャルネットワーク，知識グラフなど多岐にわたる．ego-Facebook, WebGoogle, WebTalk, CitPatents および Wikidata を除くデータセットは [12], [13] の著者らにより提供された．Wikidata は，Wikidata[2]から抽出した URI を節点としたグラフである．ego-Facebook，WebGoogle，WikiTals,

2：https://www.wikidata.org/

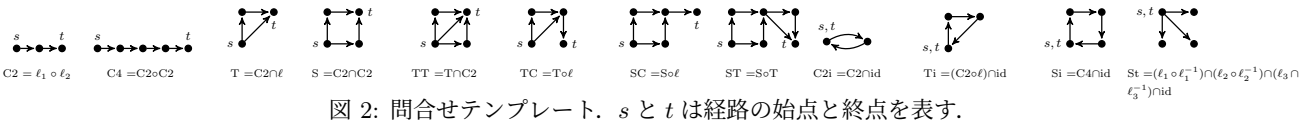


図 2: 問合せテンプレート．s と t は経路の始点と終点を表す．

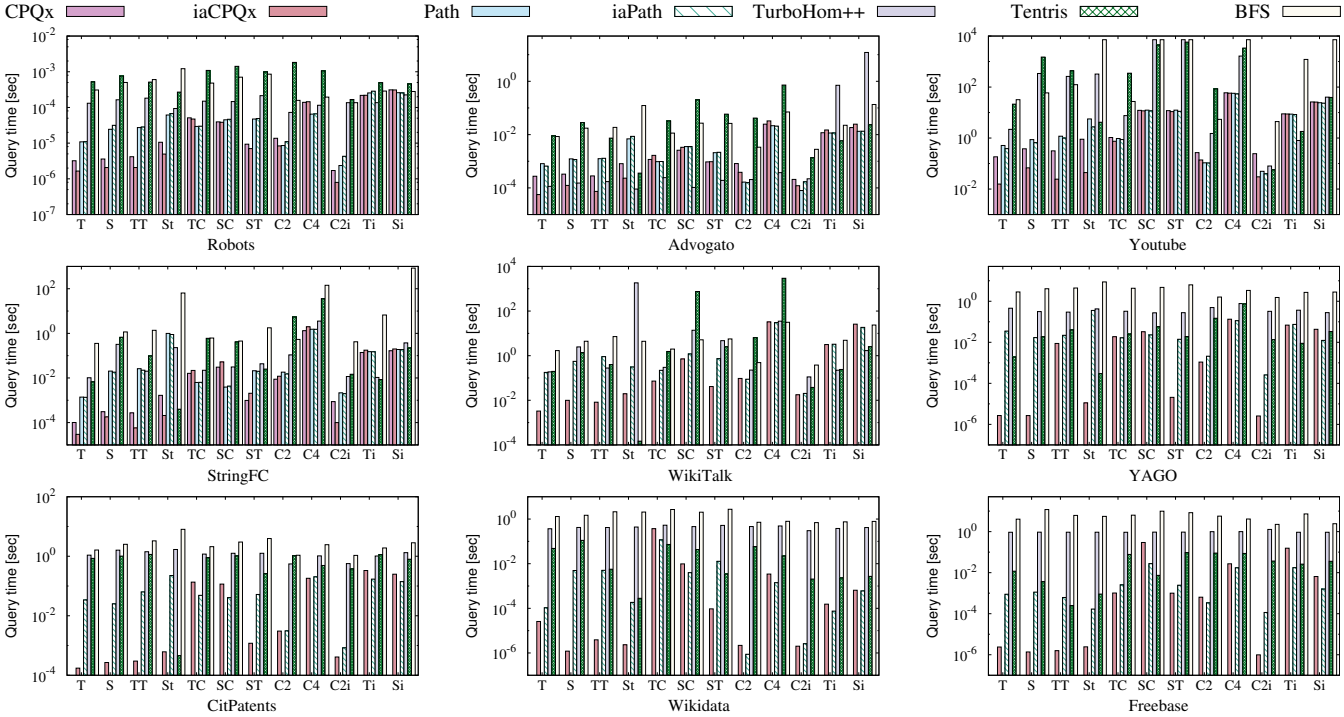


図 3: 平均問合せ時間．2 時間以内に終了しなかった問合せは打ち切りとする．CPQx および Path において WebGoogle, WikiTalk, YAGO, Wikidata, Freebase はメモリ不足により索引の構築できない．

および CitPatents は SNAP より取得できる [3]．ラベルがないグラフにおいては，YAGO のラベル分布に従うように $\lambda = 0.5$ の指数分布によりラベルを割り当てる．

問合せ．図 2 に示す 12 種類の問合せテンプレートを用いる．これらの問合せの構造は，実問合せログに頻繁に出現するものに対応している [2]．Chain，Triangle，Square，および Star に対して，それぞれ C，T，S，および St を略語として用いる．

それぞれの問合せテンプレートとデータセットに対して，ランダムにラベルを割り当てた問合せをそれぞれ 10 個ずつ作成した．このとき，長さ 2 の経路は必ずグラフ内に含まれるものを選んでいるが，問合せそのものは空の可能性がある．検索結果が空の問合せと検索結果が空ではない問合せの問合せ時間を評価するために，Yago，Wikidata，および Freebase の問合せは 50% が検索結果が空ではない問合せ，50% が検索結果が空の問合せとなるようにした．他のデータセットの問合せにおいては，多くの問合せの検索結果は空ではない．実験結果として，それぞれの問合せテンプレートに対する平均問合せ時間を示す．

比較手法．本実験では手法として 4 章で提案した CPQx; 5 章で提案した iaCPQx; 言語に特化していない構造索引 **Path** [11]; **Path** において興味ラベル系列のみを格納する構造索引 **iaPath**; 準同型サブグラフマッチング **TurboHom++** [10]; RDF エンジン **Tentris** [9]; および，幅優先探索アルゴリズム **BFS** [1] を用いる．

k はデフォルトで 2 とし，興味基づく索引では，それぞれのデータセットにおける問合せの全てのラベル系列を興味ラベル系列とする．ラベル系列が 2 より長い場合，先頭から 2 つと残りに分ける．

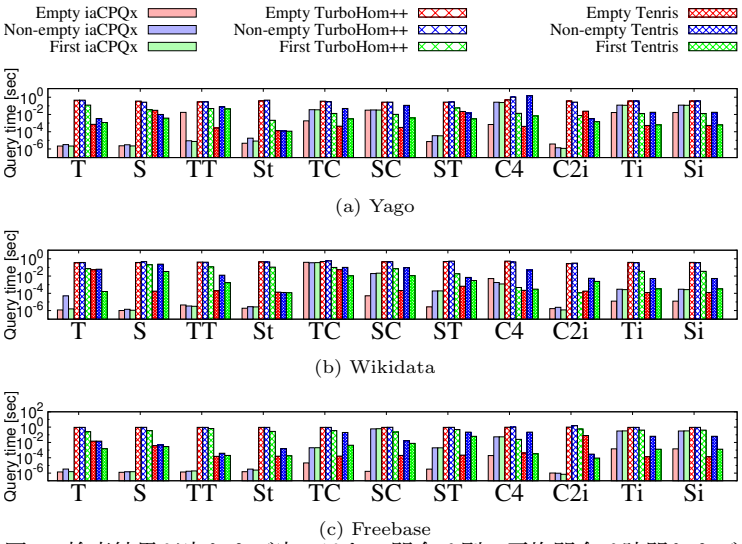
6.1 効 率 性

図 3 にそれぞれの手法の問合せ時間を示す．ページ数の都合上 9 つのデータにおける結果のみとなる．CPQx は，4.4 章で述べたように連言処理を高速化することができるため，T, S, TT, および St の問合せ時間を大きく削減していることがわかる．連言処理のみで構築される問合せでは，他の手法と比較して最大で数千倍の高速化を実現している．TC，SC，および ST のような連言処理と結合処理の両方を含む問合せでは，どちらの処理が重いかにより優位な手法が異なっている．C2，C4，Ti，および Si のような連言処理がなく結合処理を伴う問合せでは，CPQx では，I_{l2h} と I_{h2p} への両方アクセスが必要となることにより，経路索引より問合せが遅いが，大きな差はない．C2i の問合せ時間が C2 よりも早いのは，検索結果に含まれる経路数が減ることにより，発見した経路を問合せ結果に挿入するコストが減るためである．Ti および Si では，TurboHom++がいくつかのデータセットにおいて高速である．

3：http://snap.stanford.edu/

図 4: 問合せ S 評価時の CPQx と iaCPQx の history 識別子数と iaPath における経路数

Dataset	CPQx	iaCPQx	iaPath
Robots	0.4K	0.13K	2.4K
ego-Facebook	22K	19K	23K
Youtube	18M	2.0M	21M
Epinions	715K	222K	1.8M
Advogato	38.0K	6.4K	93.6K
BioGrid	275K	71.2K	499K
StringHS	49.7K	11.2K	750K
StringFC	33.3K	3.7K	388K
Yago	-	75	967K
WikiTalk	-	915K	19M
WebGoogle	-	287K	760K
CitPatents	-	36K	1.1M
Wikidata	-	3	286M
Freebase	-	8.6	79K



(a) Yago

(b) Wikidata

(c) Freebase

図 5: 検索結果が空および空ではない問合せ別の平均問合せ時間および空ではない問合せにおいて最初の結果を発見までの経過時間

表 2: 索引サイズ (IS), 索引時間 (IT), 索引構築にかかるメモリ量 (IM). "-" はメモリ不足を示す.

Dataset	CPQx			iaCPQx			Path			iaPath		
	IS [B]	IT [s]	IM [B]	IS [B]	IT [s]	IM [B]	IS [B]	IT [s]	IM [B]	IS [B]	IT [s]	IM [B]
Robots	1.78M	0.26	84.0M	0.46M	0.056	19.4M	2.0M	0.085	35.0M	0.52M	0.043	19.9M
ego-Facebook	97.4M	18.3	2.1G	15.4M	2.0	322.2M	116.9M	6.0	974.3M	20.9M	131.5	324.1M
Youtube	27.6G	57,653	389.6G	2.3G	8,116	103.3G	34.0G	28,870	195.5G	8.7G	7,283	107.2G
Advogato	56.7M	11.9	1.8G	20.5M	8.53	752M	74.4M	3.5	808M	29.0M	2.3	754M
StringHS	1.4G	521.4	71.0G	1.3G	335.9	46.0G	6.0G	260.4	42.3G	2.8G	237.3	45.3G
StringFC	1.0G	852.7	60.4G	0.97G	664.7	37.7G	5.1G	596.0	36.6G	2.3G	579.8	37.5G
BioGrid	1.7G	495.1	44.9G	0.40G	86.6	10.1G	2.48G	137.0	19.6G	0.63G	47.9	10.1G
Epinions	3.5G	849.6	84.7G	1.0G	229.6	25.3G	4.5G	264.1	36.6G	1.3G	122.8	25.3G
WebGoogle	-	-	-	4.7G	724.5	93.8G	-	-	-	5.2G	444.7	94.1G
WikiTalk	-	-	-	14.4G	3,064	286.2G	-	-	-	16.0G	1,060	278.6G
YAGO	-	-	-	3.8G	809.4	68.8G	-	-	-	3.9G	589.7	67.6G
CitPatents	-	-	-	2.2G	469.5	45.5G	-	-	-	2.5G	226.4	46.5G
Wikidata	-	-	-	1.44G	267.5	29.4G	-	-	-	1.47G	70.4	33.0G
Freebase	-	-	-	1.0G	5,696	48.8G	-	-	-	3.7G	5,176	53.1G

これは，TurboHom++は結合する前に閉路かどうか確認するのに対して，構造索引と経路索引では，結合後に確認するためである．Tentris は St において高速な場合があり，問合せの経路評価時に次数の確認とクエリ最適化が動作してるためである．CPQx，TurboHom++，および Tentris を比較すると，CPQx は多くの問合せテンプレートに対して大きく高速化している．

iaCPQx においては，経路数と history 識別子が減っていることにより，問合せ処理を高速化できている．特に，C2i においては，LookUp 処理数が大きく減っているため，問合せ時間も大きく削減できている．興味に基づく経路索引は，ラベル系列に対する経路数は元の経路索引と同じであるため，高速化の効果はない．

枝刈り性能．CPQx が経路索引に比べて高速となる理由を示す．表 4 は，問合せ S を実行する際の，CPQx と興味に基づく CPQx の history 識別子および興味に基づく経路索引の経路の平均数を示す．数が小さい場合，より無駄な比較を削減できており，枝刈り性能が高いことを示す．CPQx の history 識別子数は，経路索引と比べて非常に小さい．この実験結果より，k-path-bisimulation に基づいた経路の分割が CPQ の実行に有効であり，高速化可能であることを示している．

空の問合せの影響．ここでは，検索結果が空の問合せと空ではない問合せの実行時間を比較する．図 5 は，Yago，Wikidata，および Freebase における興味に基づく CPQx と TurboHom++ の問合せ時間を示す．より細かく TurboHom++と提案技術の問合せ時間を比較するために，最初の経路が見つかった時点で問合せを終了した場合の問合せ時間も示す．この実験結果より，提案技術が TurboHom++より，問合せ結果の空/空ではないに限らず高速であることを示している．特に，連言を含む問合せでは，有意に高速である．提案技術は，空の問合せの場合に高速であることが多いのは，(1) 空の問合せの場合は，問合せ結果の挿入コストが必要ない，および (2) 途中結果が空となることにより問合せが即座に中止することがあるためである．一

図 6: CPQx のメンテナンス時間

Dataset	Edge deletion	Edge insertion
Robots	0.0008 [s]	0.0005 [s]
Advogato	0.005 [s]	0.001 [s]
BioGrid	0.6 [s]	0.2 [s]
StringHS	0.3 [s]	0.1 [s]
StringFC	0.2 [s]	0.06 [s]
Youtube	0.9 [s]	0.3 [s]

図 7: iaCPQx のメンテナンス時間

Dataset	Edge deletion	Edge insertion	Label sequence deletion	Label sequence insertion
Robots	0.2 [msec]	0.1 [msec]	0.2 [μsec]	0.01 [sec]
Youtube	1.2 [sec]	1.1 [sec]	0.5 [μsec]	255.5 [sec]
YAGO	0.7 [sec]	0.04 [sec]	0.8 [μsec]	30.3 [sec]
Wikidata	0.7 [sec]	0.4 [sec]	1.0 [μsec]	24.2 [sec]
Freebase	1.7 [sec]	0.2 [sec]	1.1 [μsec]	21.8 [sec]

方で，空の問合せが空ではない問合せより遅い場合においては，最終結果が空であっても途中結果の経路数が多い可能性があるため，その場合空の問合せの実行時間が長くなることがある．

6.2 索引サイズ

表 2 は索引サイズ，索引構築時間，および構築時のメモリ使用量を示す．まず，CPQx は経路索引よりもサイズが小さい．これは，CPQx は一つの経路が索引上で一度しか格納されていない一方で経路ではラベル系列に対応して一つの経路が複数回格納されているためである．iaCPQx は，興味ラベル系列に含まれる経路しか格納しないため，構造索引と比較して，大きくサイズを削減できている．WikiTalk，WebGoogle，Yago，Wikidata，および Freebase では，CPQx はメモリ使用量が多く構築できていない．iaCPQx は，大規模なグラフにおいても，興味ラベル系列を調整することによりスケーラブルに構築することができ，特にラベルの偏りが大きい場合に効果的である．

CPQx の索引構築時間とメモリ使用量は経路索引より大きいが，グラフサイズに線形で増加している．iaCPQx は，索引構築時間とメモリ使用量ともに削減できる．

6.3 メンテナンス

構造索引はグラフが更新に対応して更新される．加えて，興味に基づく構造索引は興味ラベル系列の変化にも応じて更新される．索引更新の効率性検証のために，枝および興味ラベルの削除/追加に対しての更新時間を調査する．枝の更新においては 100 本の枝，興味ラベルにおいては C2 問合せのラベル系列に対して，削除と追加を行い，平均時間を測定する．

表 6 と 7 は，CPQx と iaCPQx の更新時間をそれぞれ示す．両方の索引において，索引構築時間と比較して更新時間は小さい．索引の更新は更新対象となる枝の次数が大きく影響するため，平均次数が大きい Youtube は更新時間が他と比べて大きくなっている．iaCPQx は，索引サイズがより小さいため，更新時間も小さくなっている．興味ラベル系列の変化にも索引構築時間と比べて小さい時間で更新できるが，枝の更新と比較して広範囲に影響が及ぶため枝の更新に伴う索引の更新よりも大きな時間がかかる．

7 おわりに

CPQ に特化した構造索引技術を提案した．構造索引 CPQx の設計に加えて，効率的な索引構築，索引を用いた問合せ処理，さらにメンテナンス技術を開発し，さらに興味ラベルに基づく iaCPQx を開発した．実グラフを用いた実験により，提案技術が既存技術と比べて，索引サイズを増加させずに，最大で数千倍高速化できることを示した．

謝辞 本研究は JSPS 科研費 JP20H00583 の支援によって行われた．ここに記して謝意を表す．

文　献

[1] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. *Querying Graphs*. Morgan & Claypool, 2018.

[2] Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *The VLDB Journal*, pp. 655–679, 2020.

[3] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, pp. 436–445, 1997.

[4] George Fletcher, Marc Gyssens, Dirk Leinders, Jan Van den Bussche, Dirk Van Gucht, and Stijn Vansummeren. Similarity and bisimilarity notions appropriate for characterizing indistinguishability in fragments of the calculus of relations. *Journal of Logic and Computation*, Vol. 25, No. 3, pp. 549–580, 2015.

[5] Yongming Luo, George Fletcher, Jan Hidders, Yuqing Wu, and Paul De Bra. External memory k-bisimulation reduction of big graphs. In *CIKM*, pp. 919–928, 2013.

[6] Tova Milo and Dan Suciu. Index structures for path expressions. In *ICDT*, pp. 277–295, 1999.

[7] Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, and Ehud Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *ICDE*, pp. 129–140, 2002.

[8] George Fletcher, Dirk Van Gucht, Yuqing Wu, Marc Gyssens, Sofía Brenes, and Jan Paredaens. A methodology for coupling fragments of XPath with structural indexes for XML documents. *Information Systems*, Vol. 34, No. 7, pp. 657–670, 2009.

[9] Alexander Bigerl, Felix Conrads, Charlotte Behning, Mohamed Ahmed Sherif, Muhammad Saleem, and Axel-Cyrille Ngonga Ngomo. Tentris–a tensor-based triple store. In *ISWC*, pp. 56–73, 2020.

[10] Jinha Kim, Hyungyu Shin, Wook-Shin Han, Sungpack Hong, and Hassan Chafi. Taming subgraph isomorphism for RDF query processing. *PVLDB*, Vol. 8, No. 11, 2015.

[11] George Fletcher, Jeroen Peters, and Alexandra Poulovassilis. Efficient regular path query evaluation using path indexes. In *EDBT*, pp. 636–639, 2016.

[12] Lucien D J Valstar, George Fletcher, and Yuichi Yoshida. Landmark indexing for evaluation of label-constrained reachability queries. In *SIGMOD*, pp. 345–358, 2017.

[13] You Peng, Ying Zhang, Xuemin Lin, Lu Qin, and Wenjie Zhang. Answering billion-scale label-constrained reachability queries within microsecond. *PVLDB*, Vol. 13, No. 6, pp. 812–825, 2020.