# 非巡回グラフのための拡張レンジラベリング手法

鳥井 修† 白井 智† 金井 達徳†

†(株) 東芝 研究開発センター コンピュータ・ネットワークラボラトリー

**あらまし** レンジラベリングは, XML 文書などツリーで表現されるデータの各ノードに, レンジと呼ぶツリーの位相情報を表現する数字の組を付与する手続きのことであり, レンジを用いることで XML 文書などへの問い合わせ処理の効率的な実現が可能となる. しかしながら, 従来のレンジラベリング手法は, ツリーしか考慮していない. 本論文では, 従来手法を非巡回グラフ向けに拡張し, より一般的なデータ構造への問い合わせ処理を効率的に行うことへのニーズの高まりに答える. 第 1 の拡張である「重なりレンジラベリング」は, レンジが部分的に重なりを持つことを許容することで, ツリーよりも一般的なデータ構造の位相情報をレンジによって表現することを可能にする. 第 2 の拡張である「グラフの多次元分割」は, 非巡回グラフの各ノードが複数のレンジを保持することを許容し, 複数の部分グラフへのラベル付けに分割することで, 非巡回グラフへのラベル付けを可能にする. 従来の拡張手法と比較して, 第 1 の拡張は, 主にレンジの総数を減少させ, 第 2 の拡張は, 主に各ノードのレンジ数を平滑化させ, 2 種類の拡張の組み合わせによって, 効率的な検索処理が実現されることが, 計算機実験によって検証された.

**キーワード** 非巡回グラフ, XML, レンジラベリング, 問い合わせ処理

## 1. はじめに

XML [2] 文書は, 近年, 電子化された情報の標準的なフォーマットになりつつあり, 様々な情報が XML 文書形式で記述, 交換, 保存されている. このため, XML 文書の問い合わせ処理を効率的に行うことは, 膨大な情報が溢れている今日において, 目的に合った情報を獲得する際に必須である.

レンジラベリング [5] は, 第 2 章で詳細説明を行う通り, ツリーで表現されるデータにおいて, レンジと呼ぶ始点と終点からなる数字の組を, 各ノードに 1 組ずつ付与する手続きのことである. レンジラベリングによって各ノードに付与されるレンジはツリーの位相情報を表現しており, 任意の 2 個のノードの先祖子孫関係は, これら 2 ノードに付与されたレンジのみを比較することで判定可能である. このため, レンジラベリングの結果を利用すると, 多くの種類のツリーノード検索を高速に実行することが可能である.

XML 文書はツリーとみなせるので, レンジラベリングを行えば, 要素間の先祖子孫関係を高速に判定することが可能である. このため, レンジラベリングは XML 文書の問い合わせ処理の効率的な実現に必要な基盤技術の 1 つである.

しかしながら, 従来のレンジラベリング手法はツリーしか考慮していないため, 以下の理由で十分とは言い難い.

第 1 に, 問い合わせによっては XML 文書をツリーとみなした場合には問い合わせ処理を行う際に扱いづらく, グラフとみなす必要がある点が挙げられる. 第 3.1 節で詳細説明を行う XMark [6] は, インターネットオークションをモデルとしたデータを XML 文書で表現したものである. XMark では, XML 文書エレメント間の親子関係 (例えば, 商品と商品名の関係) に加えて, XML 文書エレメント間の参照被参照関係 (例えば, オークションと落札者の関係) が頻繁に問い合わされる. このため, エレメント間の親子関係を表す有向エッジから構成されるツリーに, エレメント間の参照被参照関係を表す有向エッジを追加して, 全体として XML 文書を有向グラフと見なすことによって, 初めて様々な種類の問い合わせ処理を効率よく処理することが可能である.

第 2 に, XML 文書が標準的なフォーマットになった今日であっても, XML 文書やツリーでは表現しにくく, グラフによってのみ適切に表現可能な情報が数多く残っている点が挙げられる. 第 3.1 節で詳細説明を行う ODP [1] は, ウェブディレクトリ中に含まれるカテゴリ情報を RDF 形式で表現したものである. あるカテゴリが, 複数の異なるカテゴリのサブカテゴリになっている場合が存在することから, カテゴリはグラフによってのみ適切に表現可能である.

以上のことから, グラフへの問い合わせ処理を効率的に行うための技術へのニーズは, これから益々高まってくる.

本論文では上記の状況を鑑み, グラフ, 特に非巡回グラフへの問い合わせ処理を効率的に行うことを可能にするために, 従来のレンジラベリング手法を拡張する.

第 1 の拡張として, 第 4.1 節で詳細説明する「重なりレンジラベリング」を導入する.

従来のレンジラベリング手法によって, ツリーの各ノードに 1 個ずつ付与されるレンジの中から任意の 2 個を選択すると, 一方が他方を完全に包含する, または両者が全く重なりを持たない, いずれかの関係が必ず成り立つ. ここで, レンジが部分的に重なりを保持してよいとする拡張を行えば, ツリーよりも一般的なデータ構造の位相情報をレンジによって表現することが可能となる. この方法で表現可能であるデータ構造のことを「重なりレンジラベリング可能グラフ」, 重なりレンジラベリング可能グラフにレンジを付与する手続きを「重なりレンジラベリング」と呼ぶ.

重なりレンジラベリングにおいて, ラベル保持に必要な記憶サイズのオーダーや, ノード間の先祖子孫関係を判定する方法 (計算時間のオーダー) は, 従来のレンジラベリングと同一である. 結果として, 重なりレンジラベリングの導入により, 従来のレンジラベリング手法が保持する良好な性質を維持しながら, 問い合わせ処理を効率的に行うことが可能なデータ構造の範囲を, ツリ

ーから重なりレンジラベリング可能グラフまで広げることが可能である.

しかしながら, 非巡回グラフ全体の中には 重なりレンジラベリング可能グラフ以外のグラフも含まれているため, 一般の非巡回グラフへの問い合わせ処理を効率的に行うためには, 更なる拡張が必要である.

そこで第 2 の拡張として, 第 4.2 節で詳細説明する「グラフの多次元分割」を導入する. グラフの多次元分割は, 非巡回グラフを複数のグラフに分割する手続きである. 各ノードに 1 個のレンジ (1 次元レンジ) を持たせることによって任意の非巡回グラフの位相情報を表現することは不可能であるので, 第 2 の拡張では各ノードが複数のレンジ (多次元レンジ) を保持することを許容し, グラフの各ノードに最大で分割数に等しい数のレンジからなる多次元レンジを付与する.

具体的には, 第 1 に非巡回グラフの多次元分割を行い, 第 2 に個々のグラフに 1 次元レンジ付与を行い, 第 3 に個々に付与されたレンジをまとめて非巡回グラフ全体の多次元レンジとする.

第 2 の拡張の結果得られた多次元レンジは, 非巡回グラフの位相情報を表現しており, 従来のレンジラベリングと同様, 任意の 2 個のノードの先祖子孫関係はこれら 2 ノードに付与されたレンジのみを比較することで判定可能である.

非巡回グラフの各ノードに多次元レンジを付与する方法には, 本論文の方法以外にも, 第 3.3 節で詳細説明を行う Agrawal [7] の方法がある. この方法では, グラフのリーフノードからルートノードに向かってレンジの伝播を行うために, インターナルノードには複数, 場合によって膨大な数のレンジが付与されるという欠点を持つのに対して, 本論文の方法で 1 ノードに付与されるレンジ数は最大でも非巡回グラフの多次元分割次元数である.

第 5 章で詳細説明を行う通り, 「重なりレンジラベリング」の導入は, 主にグラフのノードに付与されるレンジの総数を減少させ, 「グラフの多次元分割」の導入は, 主にグラフの各ノードに付与されるレンジ数を平滑化させる効果をもたらし, 全体として拡張によって Agrawal の手法と比較して効率的な検索処理が実現されることが, 計算機実験によって検証された.

## 2. XML とレンジラベリング

XML (eXtensible Markup Language) はマークアップ言語の 1 つであり, タグを用いて階層構造を持つ情報を表現する. また, XML 文書はエレメントやテキストをノードとするツリーとして扱うことが可能であり, このツリーを DOM (Document Object Model) [3] ツリーと呼ぶ.

図 1 に XML 文書と, この XML 文書の DOM ツリーの例が示してある.

グラフの各ノードに, 以下の条件を満たすレンジと呼ぶデータを付与する手続きを考える.

**レンジ数条件:** レンジは始点, 終点の 2 個の数字の組であり, 各ノードに付与されるレンジ数は 1 個である.

**包含条件:** 任意の 2 ノード $u(u0, u1)$, $v(v0, v1)$ に対して, グラフ上で $u$ が $v$ の先祖になっている時, かつこの時に限り, レンジ $(u0, u1)$ はレンジ $(v0, v1)$ を包含する. ただしここで, 2 個のレンジ $(a, b)$, $(c, d)$ が $a < c, d < b$ の関係を満たす時, レンジ $(a, b)$ はレンジ $(c, d)$ を包含するという. また, グラフ上でノード $e$ からノード $f$ へのパスが存在する時, $e$ を $f$ の先祖 ($f$ を $e$ の子孫) と呼ぶ.

**重なり条件:** 任意の 2 個のノード $u$ $(u0, u1)$, $v$ $(v0, v1)$ に対して, レンジ $(u0, u1)$ とレンジ $(v0, v1)$ が部分的に重なりを持つことはない. ただしここで, 互いに包含関係にないノード $a$ のレンジ $(a0, a1)$ とノード $b$ のレンジ $(b0, b1)$ の両方に包含されるレンジ $(c0, c1)$ を持つノード $c$ が存在する場合, レンジ $(a0, a1)$ とレンジ $(b0, b1)$ とは部分的に重なりを持つという.

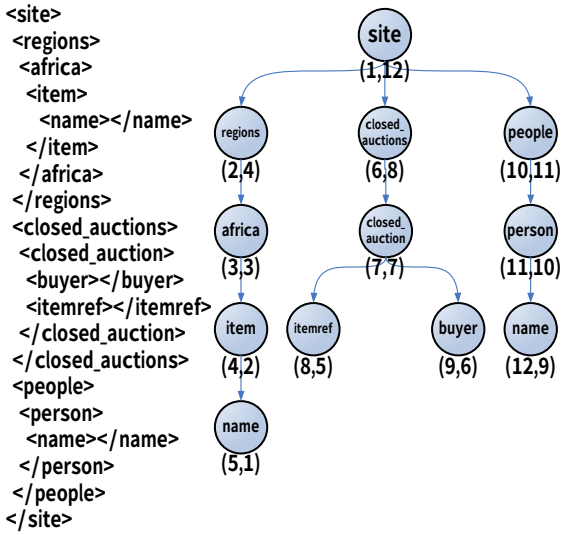


**図 1 XML と DOM ツリー**

レンジを付与する対象のグラフがツリーであれば, 上記条件をすべて満たすレンジが必ず存在するので, この時上記手続きをレンジラベリングと呼ぶ.

レンジラベリングを行う最大の目的は, 2 ノードの先祖子孫関係を高速に判定し, これを用いて様々な問い合わせを効率よく処理することである. 任意の 2 ノードの先祖子孫関係は, 上記「包含条件」から, これら 2 ノードに付与されたレンジのみを比較することで判定可能であり, 判定時間は O(1) である. これらのことから, レンジラベリングの結果を利用すると, 例えば, XPath [4] で表現される問い合わせ「A//B」など, 多くの種類のツリーノード検索を高速に実行することが可能である.

レンジラベリングを行うアルゴリズムには, ツリーのルートから深さ優先探索を行い, 各ノードの探索を開始する順番 (プレオーダー) を始点に, 各ノードの探索を終了する順番 (ポストオーダー) を終点に与える方法がある.

ツリーのノード数を $n$, エッジ数を $m$ とすると, レンジラベリングの計算時間は, 深さ優先探索に必要な計算時間, つまり O($m$), レンジに必要な記憶容量は O($n$) である.

図 1 に示した DOM ツリーの各ノードに, プレオーダー, ポストオーダーを用いたアルゴリズムでレンジを付与した結果を示してあり, 例えば先祖子孫の関係にある「closed_auction」ノードと「buyer」ノードのレンジ間に, $7 < 9, 6 < 7$ の関係が成り立っているなど, 上記 3 条件をすべて満たしている.

## 3. 非巡回グラフとレンジラベリング

### 3.1. グラフ検索の具体例

本論文では第 2 章で紹介したレンジラベリングをグラフ向けに拡張し，グラフのノード間の先祖子孫関係を高速に判定にする目的に適したラベル体系を構築する．

本節では，拡張に先立ち，拡張の必要性に関して，具体例に即して説明する．

XMark [6] は，XML 検索の代表的なベンチマークであり，インターネットオークションをモデルとしたデータを XML 文書で表現する．図 2 に，この XML 文書の一部を示した．

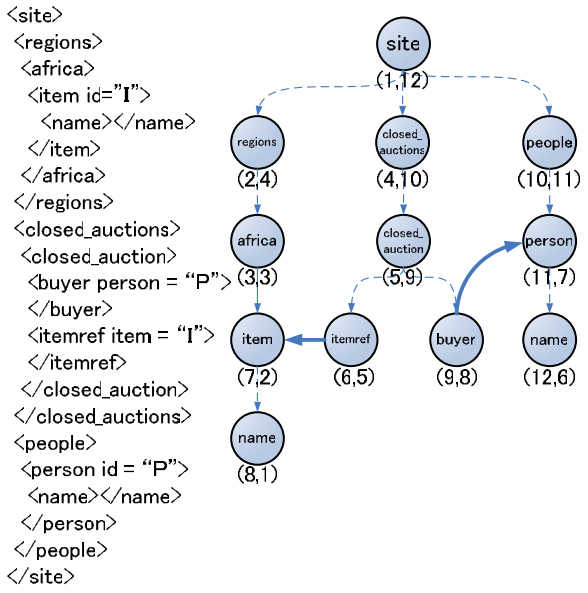


**図 2 XML 文書のグラフ表現**

このデータにおいて，「落札者が X であるオークションのリストは ?」という頻繁に行われると予想される問い合わせを処理するためには，第 1 に，closed_auction エレメントを検索し，第 2 に，値が X である name エレメントを検索し，第 3 に，第 1 の検索結果リスト中のエレメントで，第 2 の検索結果リスト中のエレメントを buyer として持つものを検索する，という手続きを行えばよい．

ここでもし，第 2 のリスト中のある name エレメントが，第 1 のリスト中のある closed_auction エレメントの buyer になっているかどうかを，これら 2 ノードに付与されたレンジのみを比較することで判定可能であれば，上記問い合わせ処理を効率よく行うことが可能である．

しかしながら，従来のレンジラベリングによって付与されるレンジを用いてこの判定を行うことは不可能である．なぜならば，XMark における個々のオークションと落札者との間には DOM ツリーにおける先祖子孫関係が一切存在しないからである．そのかわりに XMark における個々のオークションと落札者との関係は，参照元エレメント (IDREF 型の属性を保持するエレメント) から参照先エレメント (ID 型属性を保持するエレメント) への参照関係によって表現される．

よって XMark で効率的な問い合わせ処理を行うためには，XML 文書のエレメント間の親子関係を表す有向エッジ (図 2 点線) から構成されるツリー (DOM ツリー) に，エレメント間の参照被参照関係を表す有向エッジ (図 2 実線) を追加した有向グラフにおいて，ノードの先祖子孫関係を高速に判定することが求められる．これを実現するために従来のレンジラベリングを拡張する必要がある．

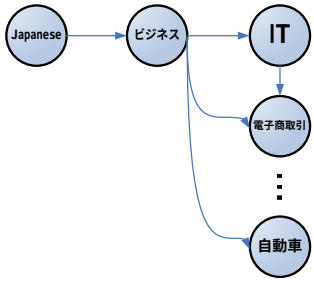


**図 3 カテゴリ**

ODP (Open Directory Project) [1] は，ボランティア方式で運営される世界最大のウェブディレクトリであり，図 3 に ODP カテゴリの一部をグラフ表現した．

一般に，あるカテゴリが複数の異なるカテゴリのサブカテゴリになっている場合が存在することから，カテゴリは，グラフによってのみ適切に表現可能である．

よって OPD で効率的な問い合わせ処理を行うためには，カテゴリを表現する有向グラフにおいて，ノードの先祖子孫関係を高速に判定することが求められる．これを実現するために，ここでも従来のレンジラベリングを拡張する必要がある．

### 3.2. 非巡回グラフ

非巡回グラフはループを持たないグラフのことである．

本論文の以下において，レンジラベリングの拡張対象は一般のグラフではなく非巡回グラフとして説明を行う．上記限定を行っても一般性を失わない理由は以下の通りである．

任意のグラフは強連結成分分解して，すべての強連結成分を縮約することで，非巡回グラフに変換することが可能である．グラフ中の 2 ノードの先祖子孫関係を判定する場合には，第 1 にこの 2 ノードが同一の強連結成分に含まれているかどうか調べ，含まれていれば 2 ノードは相互に先祖子孫の関係にあると判定する．含まれていない場合には，第 2 に非巡回グラフにおいてこの 2 ノードの先祖子孫関係を判定する．判定が困難であるのは第 2 の場合であるから，拡張は非巡回グラフに限定して議論すれば十分である．

本論文の以下において，特に混乱のない場合には，単にグラフと記述した時は非巡回グラフのことを指すものとする．

### 3.3. Agrawal の方法

従来のレンジラベリングをグラフ向けに拡張する手法の 1 つに Agrawal の方法 [7] がある．

一般に 1 次元レンジによって任意のグラフの位相情報を表現することは不可能である．そこで Agrawal の方法では，グラフの各ノードが多次元レンジを保持することを許容し，グラフから部分位相グラフを抽出して 1 次元レンジを付与し，この結果を活用して，グラフ全体の位相情報を表現する多次元レンジを付与する．ただしここで，「$G$' の任意の 2 ノード $u, v$ に対して，$u$ が $v$

の先祖である場合には $u$ は $G$ においても $v$ の先祖である」という関係が成り立つ時, $G'$ は $G$ の先祖子孫関係のうち一部を表現していることから, $G'$ を $G$ の部分位相グラフと呼ぶ.

Agrawal のアルゴリズムは, 第 1 に, グラフから部分位相グラフであるツリーを抽出して, このツリーの各ノードに従来の方法でレンジラベリングを行い, 第 2 に, グラフの子から親にレンジを再帰的に (リーフ方向からルート方向に向かって) 伝播し, 子のレンジのうち親のレンジに包含されていないもの「のみ」を親のレンジリストにマージする.

ノード数が $n$, エッジ数が $m$ であるグラフに Agrawal のアルゴリズムを適用した結果, 1 ノードに付与されるレンジ数が最大 $l'$ であるとする. ソートされている子ノードのレンジリストを伝播し, ソートされている親ノードのレンジリストにマージするために必要な計算時間は $O(l')$ であり, 結果としてソートされたレンジリストが得られる. アルゴリズムでは, エッジ数 $m$ と同じ回数だけ再帰的に伝播を行うので, 伝播に必要な計算時間は全体で $O(l'm)$ であり, Agrawal のアルゴリズムに必要な計算時間は $O(l'm)$ である. また, ラベルに必要な記憶容量は $O(l'n)$ である.

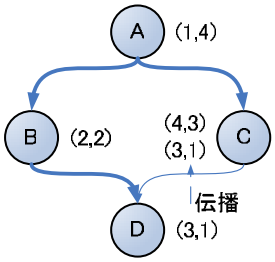


**図 4 Agrawal の方法**

図 4 に, 4 個のノードからなるグラフに Agrawal の方法によってレンジ付けを行った結果の 1 例を示す. ただし [7] では, レンジの始点として子孫ノードの最小ポストオーダーを用いているが, ここではプレオーダーを用いている. 両者に本質的な違いはない.

Agrawal の方法によって付与されたレンジは, グラフの位相情報を表現しており, 従来のレンジラベリングと同様, 任意の 2 個のノードの先祖子孫関係を, これら 2 ノードに付与されたレンジのみを比較することで判定可能である. 具体的には, 2 ノード $u$, $v$ のレンジリストを取得し, ノード $v$ のレンジリスト中のすべてのレンジが, ノード $u$ のレンジリスト中のいずれかのレンジに包含される場合には, ノード $u$ はノード $v$ の先祖であると判定, それ以外の場合 (ノード $v$ のレンジリストのレンジの中に, ノード $u$ のレンジリストのいずれのレンジにも包含されないものが存在する場合) には, ノード $u$ はノード $v$ の先祖ではないと判定する. 判定に必要な計算時間は $O(l')$ である.

Agrawal の方法によってグラフのリーフノードに付与されるレンジの個数は必ず 1 個である. しかしながら, 前述の通り Agrawal の方法はレンジの伝播を含んでいるため, インターナルノードには複数, 場合によっては膨大な数のレンジが付与されるという欠点を持つ. ノードに付与されるレンジの個数が多いと, 先祖子孫関係を判定する際に調べなければいけないレンジの個数が多くなり, 結果として検索効率の低下につながる.

図 5 に, 8 個のノードからなるグラフに Agrawal の方法でレンジを付与する具体例を示した. 伝播の結果, ノード C は 3 個ものレンジを保持している.

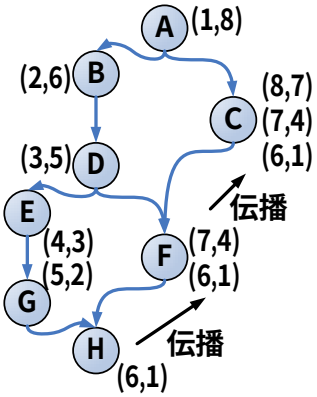


**図 5 レンジ伝播の問題点**

## 4. 重なりレンジラベリングとグラフの多次元分割

### 4.1. 重なりレンジラベリング

本節では, 第 3.2 節で紹介した Agrawal の方法の欠点を踏まえ, これと異なるレンジラベリング拡張を行う.

第 1 の拡張として, 「重なりレンジラベリング」手続きを導入する.

第 2 章で述べた 3 条件のうち重なり条件を除去し, グラフの各ノードに, 以下の条件を満たすレンジを付与する手続きを考える.

**レンジ数条件:** レンジは始点, 終点の 2 個の数字の組であり, 各ノードに付与されるレンジの数は 1 個である.

**包含条件:** 任意の 2 ノード $u(u0, u1)$, $v(v0, v1)$ に対して, $u$ が $v$ の先祖になっている時, かつこの時に限り, レンジ $(u0, u1)$ はレンジ $(v0, v1)$ を包含する.

上記手続きを「重なりレンジラベリング」と, この手続きによって位相情報が表現可能であるデータ構造を「重なりレンジラベリング可能グラフ」とそれぞれ呼ぶ. 従来のレンジラベリング手法から重なり条件を除く, つまりレンジが部分的に重なりを保持してよいとする拡張を行うことで, ツリーよりも一般的なデータ構造 (重なりレンジラベリング可能グラフ) の位相情報をレンジによって表現することが可能となる.

グラフが与えられた際, ここから重なりレンジラベリング可能グラフを抽出し, 抽出結果に重なりレンジラベリングを施すためには, グラフからツリーを抽出して従来手法で初期レンジを付与し, その後グラフ中の 2 ノード間でレンジの始点または終点をグリーディーに入れ替えることで, 初期レンジで表現されていない先祖子孫関係を順次レンジに追加する更新を行えばよい. 始点または終点の入れ替えでレンジの部分的な重なりを作り出すことで, レンジが表現する先祖子孫関係を単調増加させることができる. また, 入れ替え可能な 2 ノードが無くなるまで入れ替え手続きを繰り返すことで, グラフ中の極大な重なりレンジラベリング可能グラフ (部分位相グラフ) を抽出することができる.

アルゴリズムの詳細は以下の通りである.

第 1 に, グラフからツリーを抽出して従来手法で初期レンジ

を付与する．Agrawal の方法では，ルートから各ノードへの経路長が最長になるツリーを抽出しており，本アルゴリズムでも同じ方法を用いる．

第 2 に，ノードを始点の数字でソートしたリスト *Ls* と終点の数字でソートしたリスト *Le* の 2 種類を準備し，下記 SWAPS, SWAPE を，入れ替え可能な隣接ノードがなくなるまで繰り返す．

**SWAPS**: リスト *Ls* において隣接する 2 ノード *u*(*u0*, *u1*), *v*(*v0*, *v1*) (*u*, *v* の順番に出現する) が，
1. *v* が *u* の先祖である，
2. *u1* < *v1* である，

場合に，2 ノードの始点の値を入れ替え，*u*(*v0*, *u1*), *v*(*u0*, *v1*) とする (リスト *Ls* において *u*, *v* の順番が入れ替わる).

**SWAPE**: リスト *Le* において隣接する 2 ノード *u*(*u0*, *u1*), *v*(*v0*, *v1*) (*u*, *v* の順番に出現する) が，
1. *u* が *v* の先祖である，
2. *v0* < *u0* である，

場合に，2 ノードの終点の値を入れ替え，*u*(*u0*, *v1*), *v*(*v0*, *u1*) とする (リスト *Le* において *u*, *v* の順番が入れ替わる).

上記第 2 の手続きにおいて，始点 (または終点) が連続する 2 ノードに関して「のみ」始点 (または終点) の入れ替えを行っているので，1 度の入れ替えによって包含関係に変化が生じるのはこの 2 ノード間「のみ」である．従って，入れ替えにより，レンジによって表現される先祖子孫関係は (1 ずつ) 単調増加する．

上記アルゴリズムの実行結果は，単調性より Agrawal の方法で抽出されるツリー (部分位相グラフ) で表現される先祖子孫関係をすべて包含し，極大な重なりレンジラベリング可能グラフ (部分位相グラフ) である．

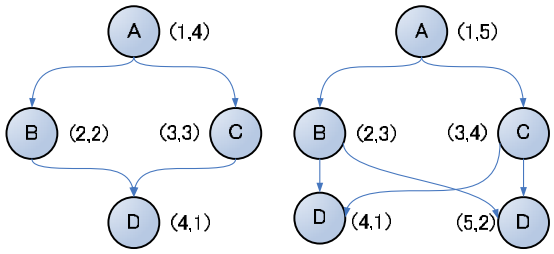


**図 6 重なりレンジラベリング**

図 6 左に 図 4 と同一の 4 個のノードからなるグラフから，図 6 右に 5 個のノードからなるグラフから，それぞれ重なりレンジラベリング可能グラフを抽出し，抽出結果に重なりレンジラベリングを行った具体例を示した．これらの具体例では，抽出前のグラフと抽出したグラフが同一になっている．

図 4 と異なり各ノードに付与されているレンジは 1 個であり，このレンジによってすべての先祖子孫関係が表現されている．上記レンジ付与は，従来のレンジラベリングでは不可能であったが，レンジの重なりを許容したことで初めて可能となった．

この例において，B (2, 2), C (3, 3), D(4, 1) の 3 個のレンジを比較すると，2 < 4, 1 < 2 より D のレンジ (4, 1) は B のレンジ (2, 2) に包含され，2 < 4, 1 < 3 より D のレンジ (4, 1) は C のレンジ (3, 3) に包含されることから，B のレンジ (2, 2) と C のレンジ (3, 3) は，共通に包含するレンジ D (4, 1) を保持し，部分的に重なりを持つことが分かる．

上記アルゴリズムは，SWAPS や SWAPE 中に 2 ノードの先祖子孫の判定を含む．この判定を高速に行うために，アルゴリズムの前処理としてグラフに Agrawal のアルゴリズムを用いて仮レンジ付与を行う．

ノード数が *n*，エッジ数が *m* であるグラフに Agrawal のアルゴリズムによってグラフにレンジ付与を行った結果，1 ノードに付与されるレンジ数が最大 *l'* であるとする．

重なりレンジラベリング可能グラフ抽出のアルゴリズムでは，前処理に $O(l'm)$ の計算時間，ツリーの抽出に $O(m)$ の計算時間が必要である．また，1 回の隣接ノード比較あたり先祖子孫判定に $O(l')$ かかり，これを $O(n^2)$ 回繰り返すことから，抽出アルゴリズム全体として計算時間は $O(l'n^2)$ である．

また，ラベル保持に必要な記憶容量は $O(n)$，抽出したグラフにおいてノード間の先祖子孫関係を判定する計算時間は $O(1)$ であり，第 2 章に示した従来のレンジラベリング手法と同一である．結果として，重なりレンジラベリングの導入により，従来のレンジラベリング手法が保持する判定時間や記憶容量に関する良好な性質を維持しながら，問い合わせ処理を効率的に行うことが可能なデータ構造の範囲を，ツリーから重なりレンジラベリング可能グラフまで広げることが可能である．

### 4.2. グラフの多次元分割

グラフの中には重なりレンジラベリング可能グラフ以外のグラフも含まれているため，一般のグラフへの問い合わせ処理を効率的に行うためには，第 1 の拡張に加えて更なる拡張が必要である．Agrawal の方法で行った拡張と同様に，本論文でも各ノードが複数のレンジを保持することを許容するが，Agrawal の方法の欠点を改善するために，伝播とは別の方法が必要である．

そこで，第 2 の拡張として「グラフの多次元分割」手続きを導入する．

グラフの多次元分割は，グラフを複数の部分位相グラフに分割し，部分位相グラフの和集合によってグラフを表現する手続きのことである．

より厳密には，ノード数 *n* のグラフ *G* が与えられた時，*k* 個のグラフ *G1*, *G2*, …, *Gk* が以下の条件を満たす時，(*G1*, *G2*, …, *Gk*) のことをグラフ *G* の多次元分割と呼ぶ．

**部分集合条件**: *G1*, *G2*, …, *Gk* のノードは，すべて *G* のノードの部分集合である．

**位相条件**: グラフ *G* の任意の 2 ノード *u*, *v* に対して，
1. *G* で *u* が *v* の先祖になっていれば，*G1* から *Gk* の少なくとも 1 個のグラフにおいて，*u* が *v* の先祖になっている．
2. *G* で *u* が *v* の先祖になっていなければ，*G1* から *Gk* のいずれのグラフにおいても，*u* は *v* の先祖になっていない．

上記 *G1*, *G2*, …, *Gk* は *G* の部分位相グラフである．グラフ *G* の部分位相グラフには様々な種類が存在するが，ツリー，重なり

レンジラベリング可能グラフなどに加えて $G$ 自身, $G$ の 1 本のエッジのみからなるグラフなども $G$ の部分位相グラフである.

グラフの多次元分割を行うアルゴリズムには, 以下の方法がある. 第 1 に, $k' = 1$ に設定する. 第 2 に, (1) $G$ の部分位相グラフ $Gk'$ を抽出する, (2) グラフの差分 $G = G - Gk'$ を計算する ($G$ の先祖子孫関係のうち, $Gk'$ に含まれていないものをすべて表現するグラフを計算し, この結果をあらためて $G$ に設定する), (3) $k'$ の値を 1 増加させる, という (1) から (3) の手順を必要回数繰り返す. 第 3 に, $Gk' = G$ に設定する. この結果得られる ($G1, G2, \ldots, Gk'$) は $G$ の多次元分割である.

上記多次元分割のアルゴリズムを用い, 任意のグラフに位相情報を表現するレンジを付与するアルゴリズムは以下の通りである.

第 1 に, グラフを多次元分割する. 第 2 に, 個々のグラフのノードにレンジを付与する. ここで $i$ 番目 ($0 < i \leqq k$) のグラフにおいて, ノード $u$ に付与されたレンジを, ノード $u$ の第 $i$ 次元のレンジと呼ぶ. 第 3 に, ノード $u$ の第 1 次元のレンジから第 $k$ 次元のレンジまでの $k$ 個をまとめてノード $u$ のレンジリストとする.

グラフの多次元分割を行うアルゴリズムは, 上記第 2 の手続き (2) に示したグラフの差分計算を含む. $G$ 中の 2 ノードが $Gk'$ において先祖子孫の関係になっているどうか判定する計算時間は O(1) であり, この判定を $G$ 中のすべての先祖子孫関係にある 2 ノードの組に対して行うため, 1 回の差分計算に必要な計算時間は $O(n^2)$ である. これを $k$ 回繰り返すことから, アルゴリズム全体でグラフの差分計算に $O(kn^2)$ の計算量が必要である. また, レンジを保持するために必要な記憶容量は $O(kn)$ である.

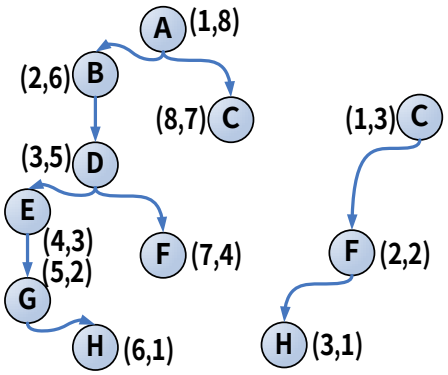


**図 7 グラフの多次元分割**

図 7 に, 図 5 と同一のグラフを 2 個のグラフに多次元分割して, レンジを付与した結果が示してある. 単一のノードが保持するレンジ数は高々 2 個である.

グラフの多次元分割を用いてノードに付与されたレンジを利用してグラフ中の 2 ノード $u, v$ の先祖子孫関係を判定する方法は以下の通りである. ノード $u, v$ のレンジリストを取得し, ノード $u$ のレンジリストが $(P1, Q1), (P2, Q2), \ldots, (Pk1, Qk1)$, ノード $v$ のレンジリストが $(R1, S1), (R2, S2), \ldots, (Rk2, Sk2)$ にそれぞれなっていたとする.

$0 < i \leqq k1, 0 < i \leqq k2$ を満たすある次元 $i$ において, ノード $v$ の第 $i$ 次元のレンジが, ノード $u$ の第 $i$ 次元のレンジに包含される場合には, ノード $u$ はノード $v$ の先祖であると判定, それ以外の場合 (すべての次元においてノード $v$ のレンジが, ノード $u$ のレンジに包含されない場合) には, ノード $u$ はノード $v$ の先祖ではないと判定する.

先祖子孫関係の判定の観点における Agrawal の方法と, グラフの多次元分割を用いた方法との類似点, 相違点は以下の通りである.

2 種類の方法とも, グラフの各ノードに複数のレンジを付与し, レンジの包含関係を用いて判定を行うという意味では類似している. しかしながら, 2 ノード $u, v$ の先祖子孫関係を判定する際, Agrawal の方法は, $v$ に付与されたレンジ「すべて」が $u$ に付与されたレンジに包含される場合に初めて先祖子孫関係が成立すると判定されるのに対して, 多次元分割を用いた方法では, $v$ に付与されたレンジのうち「1 つでも」が $u$ に付与されたレンジに包含されさえしていれば先祖子孫関係が成立すると判定される.

また, Agrawal の方法では, $v$ に付与されたレンジ $r$ が $u$ に付与されたレンジに包含されるとは, レンジ $r$ が $u$ に付与されたレンジの「いずれか 1 つ」に包含されることを意味する (判定には, 一般に複数回のレンジ比較を必要とする). このため, 判定に必要な計算時間は, ソートされているノードリストの比較に必要な時間, つまり $O(l')$ である.

これに対して, 多次元分割を用いた方法では, $v$ に付与されたレンジ $r$ が $u$ に付与されたレンジに包含されるとは, レンジ $r$ が $u$ に付与された ($r$ に対応する次元の) 「特定の」レンジに包含されることを意味する (判定は, 1 回のレンジ比較のみを必要とする).

多次元分割を用いた方法では, Agrawal の方法と異なり, 判定時に各次元 1 回のみレンジ比較を行えば十分であり, 判定に必要な計算時間は $O(k)$ である.

#### 4.3. 2 種類の拡張の組み合わせ

一般に, レンジを用いて任意のグラフの位相情報を表現するためには多次元のレンジが必要である. このため, グラフから部分位相グラフを抽出して 1 次元レンジを付与し, この結果を活用して, グラフ全体の位相情報を表現する多次元レンジを付与するという方法によって, 従来の 1 次元のレンジラベリングを多次元のレンジラベリングに拡張する.

この際, グラフから抽出する部分位相グラフをツリーにする (Agrawal の方法で採用) か, 重なりレンジラベリング可能グラフ (第 4.1 節で導入した第 1 の拡張) にするか, 2 通りの選択肢がある.

また, 1 次元レンジの結果を活用して, 多次元レンジを付与する方法には, 伝播を用いる方法 (Agrawal の方法で採用) と, グラフの多次元分割を用いる方法 (第 4.2 節で導入した第 2 の拡張) の 2 通りの選択肢がある.

表 1 に, これら 2 種類の拡張の組み合わせバリエーションを示した. 任意のグラフは上記 TP, TC, GP, CG の 4 種類方式のどれを用いてもラベル付け可能である. TP は Agrawal の方式と同一, TC, GP, GC の 3 種類が本論文独自の方式である.

ノード数が $n$, エッジ数が $m$ であるグラフに上記 4 種類の方法を用いてレンジ付与した時, 1 ノードに付与されるレンジ数の

最大数が, TP では $l'$, GP では $l''$, TC では $k'$, GC では $k''$ であるとする. 第 4.1 節で説明した単調性より, これら最大レンジ数の間には, $l'' \leqq l', k'' \leqq k'$ の関係が必ず成り立つ.

| | | 多次元化方式 | |
|---|---|---|---|
| | | 伝播 | 多次元分割 |
| 部分位相グラフ | ツリー | **(TP)** | **TC** |
| | 重なりレンジラベリング可能グラフ | **GP** | **GC** |

**表 1 拡張レンジラベリング**

ノードに付与されたレンジを用いてグラフ中の 2 ノード間の先祖子孫関係を判定するために必要な計算時間は, TP では $O(l')$, GP では $O(l'')$, TC では $O(k')$, GC では $O(k'')$ である. また, レンジを格納するために必要な記憶容量は, TP では $O(l'n)$, GP では $O(l''n)$, TC では $O(k'n)$, GC では $O(k''n)$ である.

さらに, レンジを付与するために必要な計算時間の観点から比較すると以下の通りである.

TP に必要な計算時間は, 第 3.3 節で示した通り $O(l'm)$ である. GP では, 第 4.1 節に示した通り重なりレンジラベリング可能グラフの抽出に $O(l'n^2)$, 第 3.3 節に示した通りレンジの伝播に $O(l''m)$ の計算時間がかかるため, アルゴリズム全体では $O(l'n^2 + l''m)$ の計算時間がかかる. TC では, ツリーの抽出に $O(k'm)$, 第 4.2 節で示した通りグラフの差分計算に $O(k'n^2)$ の計算時間がかかるため, アルゴリズム全体では $O(k'n^2)$ の計算時間がかかる. GC では, $O(l'n^2)$ の計算時間がかかる重なりレンジラベリング可能グラフの抽出と $O(n^2)$ の計算時間がかかるグラフの差分計算をそれぞれ $k''$ 回ずつ行うため, アルゴリズム全体では $O(k''l'n^2)$ の計算時間がかかる.

TP とその他のアルゴリズムを比較すると, $k'' \leqq k'$, $l'' \leqq l'$ の関係に加えて $k' < l''$ であることが期待されるため, 記憶容量 (付与レンジ数) や, 検索に必要な計算時間 (レンジ比較回数) の観点から, 本論文で導入した方式は TP を改善すると期待される. ただし, レンジ付与に必要な計算時間の観点からは TP が最適である.

## 5. 計算機実験

### 5.1. データ準備

本計算機実験では, ODP のカテゴリデータを利用する. カテゴリデータは「Top」をルートとする 138 万強のノードを保持するグラフであるが, 本計算機実験では, 以下に示す 2 種類の部分グラフを用いる.

1. ノード「Top/World/Japanese/アート」をルートとし, このノードから到達可能なすべてのノードからなる部分グラフ.
2. ノード「Top/World/Japanese/ビジネス」をルートとし, このノードから到達可能なすべてのノードからなる部分グラフ.

ただし, これら 2 種類のサブグラフは, いずれもループを含んでいるので, 強連結成分分解を行った上で, 個々の強連結成分を縮約し, 非巡回グラフに変換する前処理を行う.

| | ノード数 (変換前) | エッジ数 | 平均入次数 |
|---|---|---|---|
| **アート** | 1450 (1457) | 1968 | 1.36 |
| **ビジネス** | 2293 (2298) | 3094 | 1.35 |

**図 8 実験に用いるカテゴリデータ**

図 8 に, 実験に用いるグラフのノード数, 強連結成分の縮約を行う前のノード数, エッジ数, 平均入次数が記載してある. 平均入次数は, エッジ数をノード数で割ったものであり, 1 ノード平均何個の親ノードを持つかを表す. ツリーはこの値が約 1 であることから, どの程度ツリーに近いデータであるかを示す指標の 1 つである.

既存の拡張方法である TP と比較して, 本論文で導入した重なりレンジラベリングやグラフの多次元分割を用いた方法である GP, TC, GC がどの程度 TP における問題点を改善するものであるか, 実データを用いて見積もる.

### 5.2. 付与レンジ数

| アート | 合計レンジ数 | 平均レンジ数 | 最大レンジ数 |
|---|---|---|---|
| **TP** | 2196 | 1.51 | 83 |
| **GP** | 2089 | 1.44 | 53 |
| **TC** | 3489 | 2.41 | 9 |
| **GC** | 2416 | 1.67 | 6 |

| | 合計レンジ数 | 平均レンジ数 | 最大レンジ数 |
|---|---|---|---|
| **TP** | 3474 | 1.52 | 40 |
| **GP** | 3202 | 1.40 | 32 |
| **TC** | 4973 | 2.17 | 11 |
| **GC** | 3703 | 1.61 | 5 |

**表 2 合計・平均・最大レンジ数**

表 2 に, TP, GP, TC, GC を用いて 2 種類のグラフにレンジを付与した場合に, グラフ全体に付与される合計レンジ数, 1 ノードに付与される平均レンジ数, 1 ノードに付与される最大レンジ数を示した. TC, GC における最大レンジ数は, グラフの多次元分割における分割次元数に等しくなっている.

第 1 に合計 (平均) レンジ数の比較を行う. TP と GP を比較すると, TP に比べて GP のレンジ数が必ず小さくなる. 2 種類の実データに適用したところ, それぞれ 4.9%, 7.8% レンジ数減少を実現した. しかしながら, TC は TP に比べて 59%, 43% レンジ数が増加, GC は TP に比べて 10.0%, 6.6% レンジ数が増加している.

第 2 に最大レンジ数の比較を行う. 実験の結果, GP, TC, GC すべての場合に最大レンジ数が減少している. 減少幅は, GC が 93%, 88% と一番大きく, TC の 89%, 72%, GP の 36%, 20 % と続く.

### 5.3. レンジ比較回数

レンジ比較回数の比較実験を以下の方法で行った. 2 種類のカテゴリのそれぞれについて, グラフ中すべてのノードを, (1) ルートノードからから 2 本以下の枝を辿ることで到達可能なノード集合 $U$ と, (2) それ以外のノード集合 $V$ に分類する. ノード集合 $U$ 中の 1 個のノード $u$, ノード集合 $V$ 中の 1 個のノード

$v$ のすべての組合せに対して，$u$, $v$ が先祖子孫の関係になっているかどうか判定を行い，判定に必要なレンジ比較回数をカウントする．

| | TP | GC |
|---|---|---|
| アート | 1.56 | 1.24 |
| ビジネス | 1.79 | 1.27 |

表 3 平均レンジ比較回数

表 3 に示したものは，1 組のノードの先祖子孫関係を判定するために，平均して何回のレンジ比較が必要であるかをまとめたものである．アートカテゴリ，ビジネスカテゴリの実験結果ともに GC の方が TP よりもレンジの比較回数が少なく抑えられる結果 (21%, 29% 削減) が得られている．

| | TP | GC |
|---|---|---|
| アート | 114 | 6 |
| ビジネス | 52 | 5 |

表 4 最大比較レンジ数

表 4 に示したものは，1 組のノードの先祖子孫関係を判定するために，最大何回のレンジ比較が必要であるかをまとめたものである．アートカテゴリ，ビジネスカテゴリの実験結果ともに GC の方が TP よりもレンジの比較回数が少なく抑えられる結果 (95%, 90% 削減) が得られている．

### 5.4. 考察

第 1 に記憶容量に注目すると，「重なりレンジラベリング」のみを用いた拡張方式である GP が最善であり，TC, GC は TP より劣っていることが分かる．これは，「重なりレンジラベリング」はレンジの全体数削減効果をもたらすのに対して，「グラフの多次元分割」は効果がないことを意味している．

第 2 に検索速度に注目すると，「重なりレンジラベリング」と「グラフの多次元分割」の 2 つを用いた拡張方式である GC が，平均検索時間の面でも，最悪検索時間の面でも最善であることが分かる．

TP と GP では判定アルゴリズムが同一であるため，付与レンジ数とレンジ比較回数が比例する (実験結果は省略) のに対して，TP と GC (又は TC) では判定アルゴリズムが異なっているため，ノードに付与されるレンジの個数の多少が，先祖子孫関係の判定時におけるレンジ数の比較数の多少 (検索時間の長短) に比例しない．検索時間の観点からは，GC の結果が最良であるが，これは 1 つには，重なりレンジラベリングがレンジの全体数を削減させる効果を持つため，結果として先祖子孫関係を判定するために必要なレンジ数が減少したからである．もう 1 つには，グラフの多次元分割がノードに付与されるレンジ数を平滑化させる効果を持つことから，結果として先祖子孫関係を判定するために必要な最悪計算時間が改善されることに加えて，グラフの多次元分割によってノードに付与されるレンジを比較する際には，どの次元のレンジ同士を比較すればよいか検索する必要がないため，平均検索時間が改善されたからである．

以上のことから，記憶容量，検索時間の両方で Agrawal の方式を改善するためには，重なりレンジラベリングのみを用いた拡張が適している．改善効果は小幅ではあるが，理論的にも，必ず良くなることが保証されており，確実な効果が得られる．また，検索時間の面で Agrawal の方式を大幅に改善するためには重なりレンジラベリングとグラフの多次元分割の両方を用いた拡張が適している．ただし，この方式は記憶容量の面で多少 Agrawal の方式より劣っている．拡張を行う際には，用途に応じて方式を使い分けるとよい．

## 6. おわりに

本論文では，非巡回グラフのための拡張レンジラベリング手法に関する研究成果を述べた．拡張のポイントは，「重なりレンジラベリング」の導入と，「グラフの多次元分解」の導入の 2 点である．ODP のデータに適用したところ，重なりレンジラベリングの導入は主に記憶領域の削減の側面から，グラフの多次元分解は主に検索高速化の観点から有効であることが確認された．今後は，ODP 以外のデータを用いて，本拡張の有効性の検証を行う予定である．

また，1 次元のレンジラベリングに関しては，レンジを用いた効率のよい検索方法の実現 [5][8][9] や，更新に強いレンジラベリング方法の実現に関して活発に研究が行われている [11]．また，多次元のレンジラベリングに関しても Agrawal の方法をベースとする形で効率のよい検索の実現方法に関する研究がなされている [10]．これに対応して，拡張レンジラベリング手法に関しても，レンジを用いた効率のよい検索方法の実現や，更新に強いレンジラベリング方法の実現に関して検討を行う必要があるが，これは今後の課題である．

## 文　献


[1] Open Directory Project, available at http://dmoz.org/.

[2] Extensible Markup Language (XML), available at http://www.w3.org/XML/.

[3] Document Object Model (DOM), available at http://www.w3.org/DOM/.

[4] XML Path Language (XPath), available at http://www.w3.org/TR/xpath.

[5] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman, On Supporting Containment Queries in Relational Database Management Systems, In Proc. SIGMOD Conference, 425-436, 2001.

[6] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, R. Busse. Xmark: A benchmark for XML data management, In Proceedings of the 28th International Conference on Very Large Databases (VLDB), 974-985, 2002.

[7] R. Agrawal, A. Borgida, H.V. Jagadish, Efficient Management of Transitive Relationships in Large Data and Knowledge Bases, In SIGMOD International Conference on Management of Data, 253-262, 1989.

[8] D. Srivastava, S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, Y. Wu, Structural joins: A primitive for effcient XML query pattern matching, In ICDE, 141-152, 2002.

[9] N. Bruno, D. Srivastava, N. Koudas, Holistic twig joins: optimal XML pattern matching, In Proc. SIGMOD Conference, 310-321, 2002.

[10] L. Chen, A. Gupta, M. E. Kurul, Efficient Algorithms for Pattern Matching on Directed Acyclic Graphs, In ICDE, 2005.

[11] 江田 毅晴, 天笠 俊之, 吉川 正俊, 植村俊亮, XML 木のための更新に強い節点ラベル付け手法, 日本データベース学会 Letters, Vol. 1, No. 1，35-38，2002.