1F5-2

部分木を利用した判決書の柔軟な検索法

A method of information retrieval for judicial decisions using the index of syntactic subtrees

山本大介[*1] 千本達也[*1] 竹内和広[*2]
Daisuke Yamamoto Tatsuya Senbon Kazuhiro Takeuchi

[*1]大阪電気通信大学大学院工学研究科
Graduate School of Engineering, Osaka Electro-Communication University

[*2]大阪電気通信大学情報通信工学部
Department of Information and Communication Engineering, Osaka Electro-Communication University

This paper proposes an indexing method in the information retrieval systems for judicial decisions, in which we should distinguish expressions that consist of syntactic dependent words from accidental co-occurrences of the words. To solve the problem of the structure-sensitive searching, we design a scheme to code a tree structure into its corresponding strings. Based on the coding scheme, we index the subtrees of the legal documents in which all of the sentences are dependency parsed, with an extension of the TRIE structure. From some evaluations of our method, we confirmed some advantages of it on the information retrieval tasks for technical documents.

1. はじめに

判決書検索では統語関係のある複数の文節からなる表現を検索することが求められる．そのような場合，テキストの構文情報を考慮することが有益であるが，テキスト中の文の構文情報を柔軟に検索するためには，木構造で表現されることが多い構文情報の中から多数の部分木を対象とした処理が必要となり，その計算コストが問題となる．本稿では，あらかじめテキスト中の文すべての係り受け解析結果を構文情報として索引付けすることにより，左記の計算コストを軽減することを試みる．具体的には，係り受け解析木の部分木をその生成過程を示す記号列で記述することにより，テキスト検索で良く用いられるトライ構造を拡張して，対象テキスト中の特定の部分木について，その出現位置や出現数を索引付けしつつ保存することを試みる．

2. 係り受け木

木構造とは, ノードの集合 V とノード間を結ぶ枝の集合 $E = \{(x, y) \mid x, y \in V\}$ からなる無閉路有向グラフである. 各ノードは一つの親ノードと任意の数の子ノードを持つ. ただし, 根ノードと呼ばれる特別なノードは親ノードを持たない. 子ノード間に順序のある木構造を順序木（ordered tree）, 順序のない木構造を無順序木（unordered tree）もしくは根付き木（rooted tree）と呼ぶ. 本研究では, 各ノードが文字列のラベルを持つ順序木を扱う. 図 1 のように, 各ノードが文節を, 各枝が文節間の係り関係を表現した木構造を係り受け木と呼ぶ. 係り受け木の部分木は, 元のテキストに関する有益な情報を持っている.

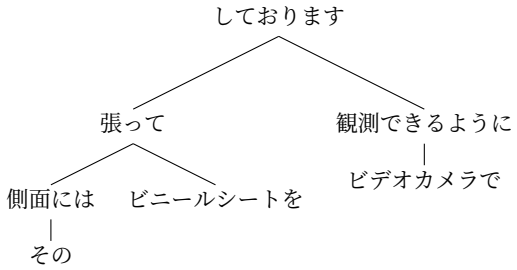

図 1 係り受け木の例

3. 順序木カーネル

Collins と Duffy と木カーネル [2] は最初の木構造間のカーネル関数であり, 構文木間の内積を与える方法として提案された. データを部分構造に再帰的に分割しそれら部分構造間のカーネル関数の和を用いて元データ間のカーネル関数を定義する, 畳み込みカーネル [3] の枠組みで設計されており, 部分構造としては次の制約を持つ部分木を用いる.

- 二つ以上のノードからなる
- 導出規則の一部だけを含まない

そして, 部分構造 s, s' 間のカーネル関数として式 1 を用いる. この関数は部分構造が等しい場合に 1, そうでない場合に 0 を返すため, 内積は共通する部分構造の数となる.

$$K_S(s, s') = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

また, 提案論文では動的計画法に基づいた効率的な計算法も提案されており, 二つの木構造 T_1 と T_2 の間の木カーネルは $O(|T_1| \cdot |T_2|)$ で計算することができる.

順序木カーネル [4] は Collins と Duffy の木カーネルから部分木に関する制約を取り払い, 部分構造として全ての部分木を認めるように一般化したものである. 本研究では, 係り受け木間の類似度として順序木カーネルを採用する. 鹿島らは二重の動的計画法を用いることでカーネル関数全体での計算量を Collins と Duffy の木カーネルと同じ $O(|T_1| \cdot |T_2|)$ のままに抑えている. しかしこのアルゴリズムは Collins と Duffy のアルゴリズムと同様に純粋に木構造の対が与えられた場合に効率的に計算することを目的としており, 類似検索においてクエリと多くのデータとの間の類似度を全て計算する場合にはデータ数に比例して計算

連絡先: 山本大介，2013 年 4 月より国立大学法人神戸大学工学研究科技術員，Email:cone@takelab.jp

量が増加してしまうことが問題である.

4. 部分木の索引化

木構造間のカーネル関数の計算は本質的には共通の部分木を数えることであるので, 検索対象となる各木構造が含む部分木を予め索引として構造化しておくことができれば, カーネル関数と同等の類似度による木構造の高速な類似検索を行うことが出来ると期待できる. ある木構造が含む全ての部分木を列挙した場合, これらの間には包含関係があるので, 各ノードに文字を割り当て根からそのノードまでの文字を繋げた文字列に関する情報を保持する索引構造であるトライ（TRIE）[1] を応用してこれらを管理することができれば, 効率的な保存・検索に役立てることができる.

4.1 木の系列表現

複数の木構造をトライに格納するために, 木構造を「直前に処理したノードの i 代前のノードの一番右の子ノードとしてノード n を追加する」操作の系列として表現する. 木構造の系列表現の例を図 2 に示す. ここで, A〜G のアルファベットはラベルである. S は系列の始点を表すためのものであり, 木構造の一部ではない. 木構造を表現する系列は全てこの S から始まるものとする. また, ラベルの前の整数が i を表している. 図中の吹き出しが示すように, 系列の根から任意の要素までの部分系列は特定の部分木を表現しており, 部分系列の包含関係は部分木の包含関係に対応している. また, 部分系列から始点 S を除いた系列の長さは表現している部分木のノード数に等しい.

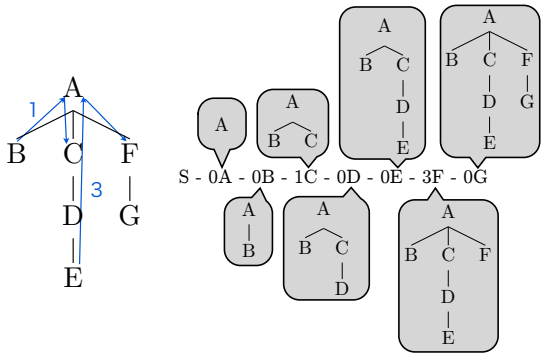

図 2　木構造の系列表現

4.2 部分木トライ

先の系列表現を利用して, ある木構造の全ての部分木の系列を統合したトライを**部分木トライ**と呼ぶ. 部分木トライの各ノードには, 系列の始点からそのノードまでの部分系列が表現する部分木の, 元の木構造における出現回数を記憶する. 部分木トライの例を図 3 と図 4 に示す. それぞれ, (a) が元の木, (b) がその木から構築された部分木トライである. 各ノードのアルファベットはラベルを表しており, ラベルの前の整数が i を表している. また, ラベルの後ろの括弧付きの数値は部分木の出現回数である.

部分木トライを利用して計算される共通部分木数は順序木カーネルによる類似度と等しい. また, 全ての部分木を列挙するのではなく, 特定の制約を満たすものだけを扱うことで他のカーネル関数に等しい類似度を計算することも可能である.

4.3 構築アルゴリズム

部分木トライを構築する際, 与えられた木構造の全ての部分木を列挙しそれぞれを系列に変換することもできるが, これらを同時に行うことができれば効率的である. そのためのアルゴリズムを以下に示す.

(a)　(b)

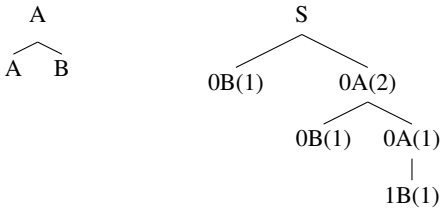

図 3　部分木トライの例 1

(a)　(b)

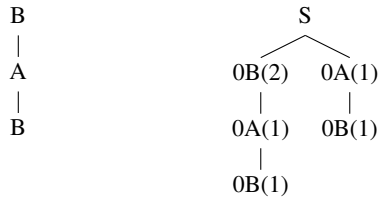

図 4　部分木トライの例 2

- 与えられた木構造のノードを深さ優先・後行順で右から左に処理する.
- ノード n を根とする全ての部分木を列挙したトライの部分（**基本パーツ**）を部分木トライに統合する. n の基本パーツは以下の手順で構築できる.
  - 基本パーツの根として,「0, n.label」を用意する.
  - 根の下に, n の各子ノードの**拡張パーツ**を接ぎ木する. ノード c の拡張パーツとは, c の基本パーツの各ノードに c の全ての右の兄弟の拡張パーツを接ぎ木したものである. 接ぎ木する際, 根ノードのみ深さ i を再計算する必要がある. c が右の兄弟を持たない場合, 基本パーツと拡張パーツは等しい.

基本パーツと拡張パーツの例を図 5 に示す. 図 5(a) の木におけるノード D の基本パーツが図 5(b), 拡張パーツが図 5(c) である. 図 5(c) で下線を付した整数は, この拡張パーツを構築する際に再計算する必要がある. D の拡張パーツにおいては, その右の兄弟である F と H 及びその子ノード G, I は出現するが, 左の兄弟である B やその子ノード C は出現しない.

(a)　(b)　(c)

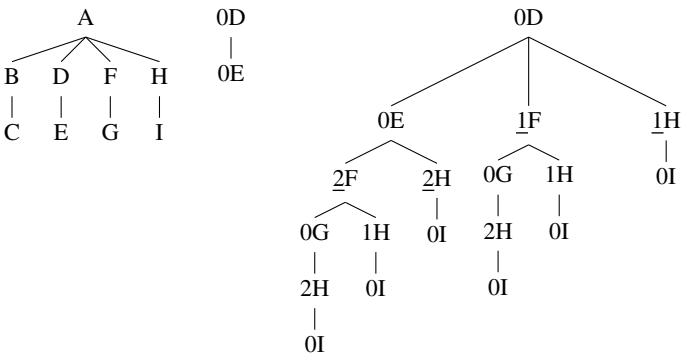

図 5　基本パーツと拡張パーツ

4.4 データベーストライ

検索対象となる全ての木構造から部分木トライを構築し, それらを一つのトライ（データベーストライ : DB トライ）に統合する. この時, 各部分木の出現回数は元の木構造毎に分けて記録する. 図 3(b) の部分木トライと, 図 4(b) の部分木トライを統合した DB トライを図 6 に示す. ただし, 括弧内は部分木のそれぞれの木での出現回数であり, x は図 3 の木を, y は図 4 の木をそれぞれ表している.

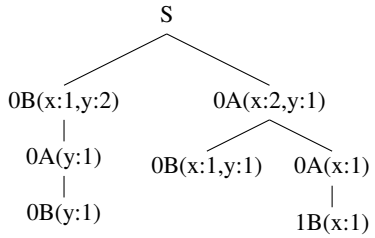

図 6　データベーストライの例

4.5 順序木カーネルの計算

クエリとなる木から, 同様の手法で部分木トライ（クエリトライ）を生成し, DB トライとマッチングしながらクエリトライを走査することで, 全ての木構造との共通部分木数を同時に計算することができる. 図 7(a) の木構造から生成した図 7(b) の部分木トライをクエリトライとし, 図 6 の DB トライを検索する場合の手順を以下に示す. 但し, 図 7(b) の丸付きの数字は処理の順番であり, 説明のために付したものである.

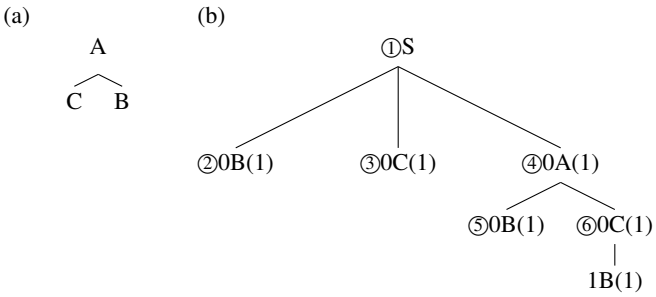

図 7　クエリトライ

1. S は必ず部分木トライの根であるので, 探す必要はない.
2. S の子から 0B を探す. DB トライの該当ノードに (x:1, y:2) とあるので保存する.
3. S の子から 0C を探す. DB トライには該当ノードがない.
4. S の子から 0A を探す. DB トライの該当ノードは (x:2, y:1). 合計で (x:3, y:3).
5. A の子から 0B を探す. DB トライの該当ノードは (x:1, y:1). 合計で (x:4, y:4).
6. A の子から 0C を探す. DB トライには該当ノードがない.

最終的に, DB トライに対するクエリトライとの共通の部分木数は, x, y ともに 4 であることが分かる. ここで, クエリトライにおける⑥0C がマッチしなかった時点で, その子ノードである 1B は処理する必要がない. これは, ⑥0C がどれだけ子孫を持っていても同じである. 計算量はクエリトライのノード数を $|Q|$ とすると $O(|Q|)$ であり, DB トライのノード数には依存しない.

5. 実験

5.1 データ

自然言語で書かれた実際の刑事裁判の判決書を句読点毎に区切ったものを使う.（本稿ではこの単位を節と呼ぶ.）これら節に形態素解析ツール CaboCha[5] を用いて機械的に係り受け解析を行い係り受け木を生成した. これら係り受け木に含まれる部分木の種類数を図 8 に示す. 部分木の種類数は係り受け木のノード数に対して指数的に増えるが, 木の構造によってばらつきが大きいことが分かる.

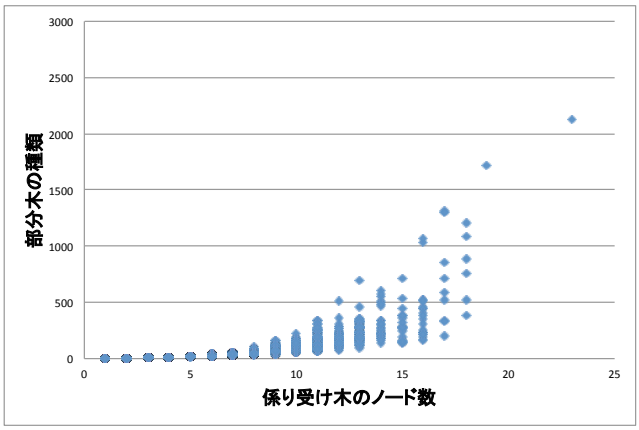

図 8　部分木の種類数

5.2 計算時間の比較

DB トライを用いた共通部分木数の計算がデータ数に依存しないことを確認するために, クエリとなる木構造と検索対象となるデータ全ての間の共通部分木数を, 順序木カーネルを用いて個別に計算した場合と DB トライを用いて一度に計算した場合とで計算時間の比較を行った.

実験はデータ数を 1,000 節から 19,000 節まで 1,000 区切りで変化させ, クエリとしてデータセット内のデータを用いた場合と, データセット外のデータを用いた場合のそれぞれについて行った. このとき, クエリとなる木構造のノード数が同じになるようにそれぞれ 500 節をクエリとして選択した. 実験結果を図 9 に示す. 計算時間は 500 のクエリについて 2 回ずつ計算した平均である.

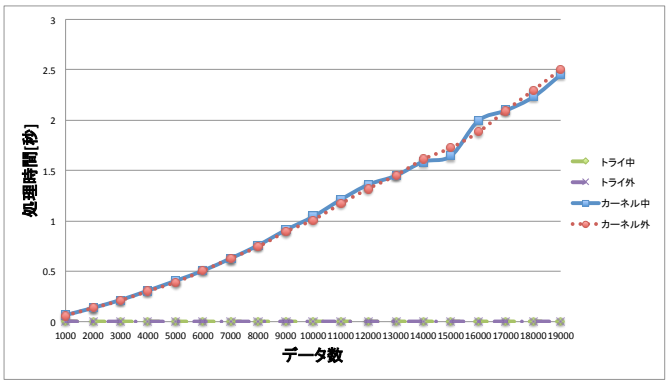

図 9　検索実験の結果

どちらのクエリを用いた場合でも, 順序木カーネルを用いて個別に計算した場合には検索対象のデータ数に応じて計算時間が

増加しているのに対して, DB トライを用いた場合にはデータ数に関わらずほぼ一定の時間で計算できていることが分かる.

5.3 部分木を利用した判決書検索

5.2 節では，本稿の提案手法が，部分木の索引化について有効に機能することを，一般的な木カーネル関数を用いて実験により示した．ここでは，部分木索引を利用した判決書検索の可能性について述べる．

DB トライを用いて特定の部分木を含むデータを検索する単純な方法は次の通りである．ここで，クエリとなる部分木の系列表現を「S0A0B」とした時，その部分木を含むデータを検索することを例に考えると，その処理は，まず DB トライの根の子ノードから「0A」を探し，次にその子ノードから「0B」を探せば良い．この手順で探し出したノードには出現情報が保存されているため，どのデータにそれぞれ何回出現するかはすぐに確認できる．つまり，単純な部分木の検索であれば，順序木カーネルの計算と同じく，この検索はクエリとなる木構造のノード数にのみ依存し，DB トライのノード数に依存しない．

この単純な手順でも，実際の判決書検索においては係り関係を持った複数文節にわたる検索に有益な柔軟性を持たせることができる．例えば,「事情を-考慮した」という部分木は，係り関係に基いて,「事情を総合的に考慮した」という単純文字列では一致しない文を検索できる．このような例の検索は，通常の検索手法ではキーワードの出現順や出現距離を考慮した拡張で対応させざるを得ず，提案手法が有用であると考える．

また，特定の接頭辞を持つ文字列を列挙することが容易であるというトライの性質は，特定の部分木を含む部分木を列挙する，という形で利用することができる．例えば，図 6 の DB トライにおいて A を根とするような部分木を列挙したい場合，まず根ノードの子ノードから「0A」を探し，次にその子孫ノードを列挙することで,「S0A」「S0A0B」「S0A0A」「S0A0A1B」の 4 種類の部分木が存在することが分かる．この性質を利用することで，提案手法は言い換え検索にも柔軟に対応できる可能性がある．

以上のように，本稿の提案手法により，検索文章中の特定の係り受け部分木を現実的な時間で抽出することが可能となり，より高度な判決書検索を実時間で実現する基礎ができたものと考える．今後の課題としては，判決書検索への応用に即して，上記のような基本的な部分木の索引化の拡張を工夫することにより，より検索の柔軟性を高めることが挙げられる．そのためには，判決書に特徴的な部分木パターンを応用に即して検討していく必要がある．特に，それら部分木パターンの検討に自動的な方法論を導入していきたい．既に我々は 5.2 節で検討した木カーネル関数を類似度として用いて，節の階層的クラスタリング実験を行っているが，その結果では，ノードに対して何の重み付けもしていない，すべての部分木を平等に扱った文の類型からは，高度な判決検索に資する拡張検索がうまく定義できていない．すなわち，クラスタリングにおいて，目的に即した類似関数の定義や，部分木の重み付けが課題であることが判明している．これを解決する方向性としては，判決書には人手で量刑事情などの付加情報をつけたデータベースが開発されていたり，複数の判決書を比較対照した研究がなされていたりするので，それらの先行データにもとづいて，人間の検索目的に応じた部分木索引の改良を進めていきたい．

6. おわりに

本稿では，テキストを単なる文字列ではなく係り受け関係を表現した木構造として捉え，文構造を感知する検索を行うための索引構造を提案・実装した．提案した実装の有効性を評価するため，実際の判決書すべての節を係り受け解析を行ったデータを対象とし，木構造間のカーネル関数関数を計算課題として，個別にカーネル関数を計算する場合と，提案実装の計算速度を比較した．その結果，提案の実装では，部分木の検索コストがデータ数に依存しないため，特定の部分木の存在を検索条件として利用する検索に，提案した部分木の索引が有益に機能することを示した．今後は，既存の判決書分析データを参考に，検索において係り受けの部分木を高速に検索できることを利用して，より高度な判決書検索システムを開発していきたい．

参考文献

[1] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, Vol. 463. ACM press New York, 1999.

[2] Michael Collins, Nigel Duffy, et al. Convolution kernels for natural language. In *In Proc. of Neural Information Processing Systems (NIPS 2001)*, Vol. 14, pp. 625–632, 2001.

[3] David Haussler. Convolution kernels on discrete structures. Technical report, UCSC-CRL-9910, UC Santa Cruz, 1999.

[4] 鹿島久嗣, 坂本比呂志, 小柳光生. 木構造データに対するカーネル関数の設計と解析. 人工知能学会論文誌 = Transactions of the Japanese Society for Artificial Intelligence : AI, Vol. 21, pp. 113–121, 2006.

[5] 工藤拓, 松本裕治. チャンキングの段階適用による日本語係り受け解析. 情報処理学会論文誌, Vol. 43, No. 6, pp. 1834–1842, 2002.