# 構文木による構文木付きコーパス検索システム

吉田 恭祐† 橋本 泰一† 徳永 健伸† 田中 穂積†

† 東京工業大学大学院 情報理工学研究科 計算工学専攻

**{rincho, taiichi, take, tanaka}@cl.cs.titech.ac.jp**

## 1 背景と目的

近年，自然言語処理の分野では，大規模な言語資源を利用した統計的手法が中心となっている．特に，構文構造付きコーパスは，統計的手法に基づく言語処理の学習データとしてだけでなく，言語学や言語処理研究の基本データとしても貴重な資源である．そのため，大規模な構文構造付きコーパスの作成が必要となるが，複雑な構造を持つ構文構造付きコーパスを全て人手により作成するのは非常に困難であり，コーパス作成支援システムに関する研究が行われるようになった．

構文構造付きコーパスの作成を支援するシステムはいくつか提案されているが [3, 10]，これらのシステムは，GUI ツールを用いて，構文構造付けをするコーパスのファイル形式や品詞ラベルの不整合を防ぐのが主な機能である．しかし，それだけでは，正しい構文構造付きコーパスの作成には不十分であり，構文構造の一貫性を保つための支援が必要となる．

構文構造の一貫性を保つための支援として，過去の事例の参照は非常に有効である．複数の構造付けのどれが正しいのかを迷った場合に，それらの構造と類似した部分を持つ構文木を参照できれば，正しい構文構造付けが容易になり，一貫性を保つことを支援できる．

構文構造付きコーパス検索システムは，grep などの文字列検索とは異なり，複雑な構造検索を行うので，検索時間の遅さが，しばしば問題とされている．既存の構文構造付きコーパス検索システム [6, 2] においても，主な問題として，検索時間の遅さが挙げられているが，現在，検索時間を高速化する優れた手法は提案されていない．今後，コーパスの規模が更に大きくなっていった場合，検索時間の高速化が重要になってくる．本論文では，検索時間の高速化を目的とした，構文構造付きコーパス検索システムを提案する．

## 2 検索システムの概要

構文構造付きコーパスのデータベース化は，吉川の手法 [8] を用いて行った．吉川は，構文構造付きコーパスと同じ木構造を持つ XML 文書をオブジェクト関係データベースを用いて格納/検索する手法を提案している．この検索システムは，クエリとして XPath を与え，XPath を SQL 文に変換し検索を行う．これを応用して，部分木として与えられたクエリを SQL 文に変換し検索を行う検索システムを構築した．

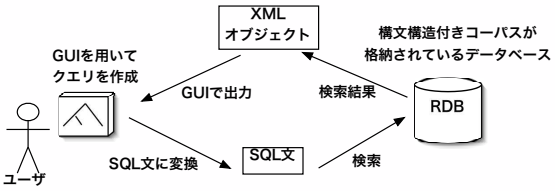


図 1: 検索システムの概要図

図 1 に提案する検索システムの概要図を示す．ユーザは，GUI を用いてクエリを作成し，検索を開始する．構文構造付きコーパスは，予め Relational Database に格納し，部分木として表現されたクエリは，SQL 文へと変換され，データベースでの検索が行われる．検索結果は，XML オブジェクトとしてデータベースから出力され，ユーザはその結果を GUI を通して木構造として確認する．

## 3 提案手法

吉川の手法は，XML 文書の検索システムの評価実験で一般的に用いられる XML 文書[1]を用いた検索実験で，検索時間の速さを示した [8]．しかし，XML 文書は DTD（Document Type Definition）で，ノードのラベルや構造が予め決められていることを前提としたもので，多様な構造を持つコーパスほど複雑な構造を持たない．よって，吉川の手法をそのまま構文構造付きコーパス検索システムに用いても，XML 文書と同等の検索時間は期待できない．

### 3.1 クエリの分割

クエリのノード数が多くなった場合，それをそのまま SQL1 文で検索する手法では，検索時間が非常に遅くなることが分かった．この問題を解決するため，ノード数の多いクエリは，検索単位と呼ぶいくつかの部分木に分割し，それぞれを別の SQL 文に変換して検索する手法を提案した [13]．この手法では，以下の決定が，検索時間の高速化にとって重要となる．

[1] シェークスピアの戯曲を Jon Bosak がタグ付けしたもの

- 検索単位のサイズ
- 検索単位の検索順序

以下，これら 2 つの決定方法について説明する．

### 3.2 検索単位のサイズ

検索単位のサイズは，SQL1 文で効率的に検索可能なノード数とし，その最大値を最適クエリノード数と呼ぶ．吉田らが [13] で提案した検索システムでは，最適クエリノード数を人手で決定していたが，それを自動決定する手法を提案する．その手順を以下に示す．

まず，検索対象コーパスから，ノード数 1〜31 のクエリを各 50 個ランダムに抽出し，SQL1 文で検索した場合の各ノード数ごとの平均検索時間をサンプリングデータとして算出する．そのデータをもとに，以下の 2 つの数式をともに満たす最小値 i を最適クエリノード数とする．n は許容可能な検索時間，m は連続数を表すパラメータである．

$$\begin{array}{l} i-m+1,\cdots,i \le n \\ i+1,\cdots,i+m > n \end{array}$$

図 2 のサンプリングデータで，n を 0.5，m を 3 とした場合，ノード数 15 が最適クエリノード数となる．

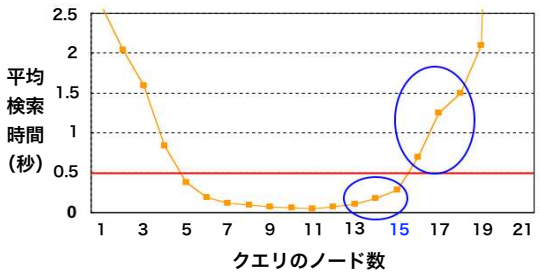


図 2: 最適クエリノード数決定の例

### 3.3 検索単位の検索順序

検索単位の検索順序は，検索単位に分割された順序と同じである．よって，クエリの分割のアルゴリズムを，図 3 の具体例を用いて説明する．説明の便宜上，最適クエリノード数を 4 とする．

まず，クエリの各ノードに規則の出現頻度を与える．図 3 の場合，*NP-SBJ* → *an Indian NP* という規則の出現頻度が 13 となる．次に，最小頻度ノードを選択する．この時点で，選択した部分木の総ノード数は 3 で（図 3(a)），最適クエリノード数に達していないので，隣接するノードで最小頻度の”NP-SBJ”を参照する．このノードを選択した場合，選択した部分木の総ノード数は 6 となり，最適クエリノード数を越えるので，選択はしない．次に，2 番目に頻度の小さい”ADJP”を参照し，この場合は選択を行い，選択した部分木の総ノード数は 4 となる（図 3(b)）．まだ，最適クエリノード数には達していないので，次に頻度の小さい”NP”を参照するが，このノードを選択すると，総ノード数が 6 となるので，選択は行わない．他に参照するノードがないので，この時点で選択されたノードより構成する部分木が検索単位となる．それ以降の分割は，検索単位に隣接するノードのうち，頻度が低いものを優先的に選択し行う（図 3(c)）．

## 4 評価実験

表 1 に示す 7 つのコーパスを用いて，3 節で述べた手法の評価実験を行った．実験は，まず 3.2 節で述べた手順で，n を 0.5，m を 3 とし，最適クエリノード数を求める．次に，サンプリングデータの算出に用いたものと同じクエリを，分割手法で検索を行い，各ノード数ごとの検索時間を測定した．

図 4，5，6，7 に示すように，SQL1 文で検索する手法では，どのコーパスにおいてもノード数が 16 以上で 1 秒を越え，更にノード数が増えると，非常に検索時間が遅くなることが分かる．一方，分割手法はノード数が多くなっても，安定した速さで検索を行っている．この 4 つのコーパスとは対照的に，図 8，9，10 に示す 3 つのコーパスでは，最適クエリノード数が 31 となり，分割の必要はないという結果となった．この要因について，5 節で考察する．

## 5 考察

検索時間に差が出る要因として，以下のものが考えられる．

1. 文数
2. 1 文の平均 node 数
3. ラベル
4. 構造の特徴

1 と 2 は，多くなればそれだけ検索に負荷がかかると思われるものだが，表 1 を見る限り，あまり関連はなさそうである．また，3 については，表 1 のラベルの異なりだけを見たのでは判断は難しいが，クエリに記述するラベル条件による絞り込みが巧く働いて，検索時間が速くなる可能性がある．そこで，クエリにラベル条件を記述せずに，クエリの構造のみを用いて検索すると，検索時間がどのように変化するかを調べる実験を行った．

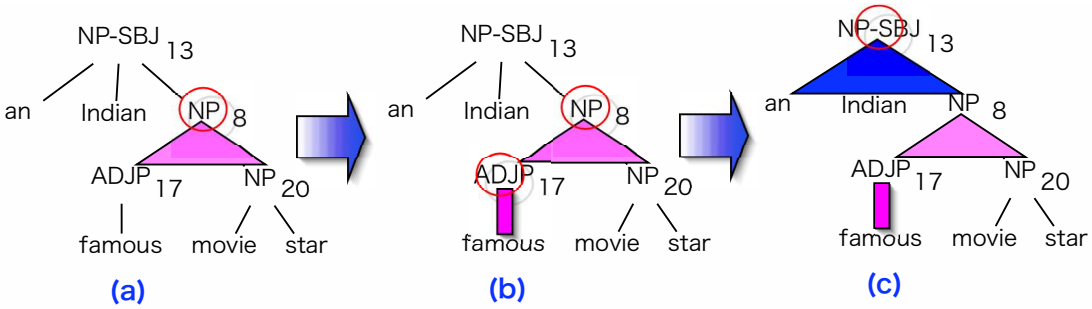


図 3: クエリ分割の例

コーパスは，Penn Treebank Corpus，東工大コーパス（RWC）を用いた．使用したクエリは，4 節で用いたクエリをノード数 2〜31 まで各 100 個用いて検索を行った．検索の際，クエリのラベル条件は用いず，構造がマッチするもの全てを検索結果とした．図 11 に，その結果を示す．

Penn Treebank の場合，ノード数 20 から非常に検索時間が遅くなっている．一方，東工大コーパスは，全てのノード数に対して安定しており，検索時間が非常に遅くなることはなかった．ラベル条件があってもなくても，検索時間に差が出るということから，要因は 4 の構造の特徴が関連していることが分かった．

そこで，構造の特徴について，いくつかの調査を行ったところ，それぞれのコーパスの平均枝数に，関連性を見出した．図 12 に，平均枝数の分布図を示す．

この調査から，SQL1 文のノード数が増えた場合に検索時間が非常に遅くなるコーパスは，平均枝数が多く，検索時間が遅くならないコーパスの平均枝数は少ないことが分かる．しかし，それらの境界は非常に近接したものであり，はっきりと断定はできない．これについては，更に詳しい調査が必要である．

## 6 まとめと今後の課題

本論文では，吉川の手法を基に関係データベースに格納した構文構造付きコーパスから構文木を検索する手法の改善方法を提案した．最適クエリノード数と規則の頻度を用いて，ノード数の多いクエリをいくつかの検索単位に分割し，それぞれを別の SQL 文で検索する手法を提案した．評価実験では，4 つのコーパスで，検索時間の高速化を示した．また，クエリのノード数が多くなったとき，コーパスによって検索時間に大きな差が出ることを見出し，その要因が，構造の特徴にあることを確認した．

今後の課題として，以下のようなものが挙げられる．

- 平均枝数と検索時間の関連性の詳しい調査
- NOT などの検索オプションの追加

## 参考文献

[1] Afonso, Susana, Eckhard Bick, Renato Haber, and Diana Santos. ”floresta sintá(c)tica”: a treebank for portuguese. In *Proceedings of LREC 2002*, pp. 1698–1703, 2002.

[2] S. Bird, Y. Chen, S. Davidson, H. Lee, and Y. Zheng. Extending xpath to support linguistic queries. In *UWA Language Science Group 2004 Program - Symposium on Speech and Language Technology*, 2004.

[3] H. Cunningham, V. Tablan, K. Bontcheva, and M. Dimitrov. Language engineering tools for collaborative corpus annotation. In *Proceedings of Corpus Linguistics 2003*, 2003.

[4] Chung hye Han, Na-Rae Han, Eon-Suk Ko, Heejong Yi, and Martha Palmer. Penn korean treebank: Development and evaluation. In *Proceedings of the 16th Pacific Asia Conference on Language, Information and Computation*, 2002.

[5] König, Esther, Lezius, and Wolfgang. The tiger language - a description language for syntax graphs, formal definition. Technical report, Institut für Maschinelle Sprachverarbeitung, University of Stuttgart, 2003.

[6] König, Esther, Lezius, Wolfgang, Voormann, and Holger. *TIGERSearch User's Manual*, 2003.

[7] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The penn treebank: Annotating predicate argument structure. In *ARPA '94*, 1994.

[8] 吉川正俊, 志村壮是, 植村俊亮. オブジェクト関係データベースを用いた XML 文書の格納と検索. 情報処理学会論文誌, Vol. 40, No. 0, 1999.

[9] Tomoya Noro, Taiichi Hashimoto, Takenobu Tokunaga, and Hozumi Tanaka. A large-scale japanese CFG derived from a syntacticaly annotated corpus and its evaluation. In *Proceedings of the 3rd Workshop on Treebanks and Linguistic Theories (TLT2004)*, pp. 115–126, 2004.

[10] Oliver Plaehn and Thorsten Brants. Annotate – an efficient interactive annotation tool. In *Proceedings of the Sixth Conference on Applied Natural Language Processing ANLP-2000*, Seattle, WA, 2000.

[11] 野呂智哉, 白井清昭, 徳永健伸, 田中穂積. 大規模日本語文法の開発ー事例研究. 情報処理学会研究報告 NL-150, pp. 149–156, 7 2002.

[12] Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. Building a large-scale annotated chinese corpus. In *Proceedings of COLING 2002*, 2002.

[13] Kyosuke Yoshida, Taiichi Hashimoto, Takenobu Tokunaga, and Hozumi Tanaka. Retrieving annotated corpora for corpus annotation. In *Proceedings of LREC 2004*, 2004.

| コーパス | 文数 | 語数<br>(終端ノード数) | 1 文の平均<br>node 数 | 1 文の平均<br>word 数 | 非終端ラベル<br>の異なり | 終端ラベル<br>の異なり | 最適クエリ<br>ノード数 |
|---|---|---|---|---|---|---|---|
| Penn Treebank Corpus[7] | 48,884 | 1,244,104 | 45.34 | 25.45 | 1225 | 44,175 | 15 |
| TIGER Corpora[5] | 40,020 | 616,414 | 39.24 | 15.40 | 723 | 73,385 | 15 |
| Penn Korean Treebank[4] | 5,083 | 102,095 | 59.48 | 20.09 | 123 | 3,317 | 15 |
| FLORESTA sintá(c)tica[1] | 8,307 | 191,476 | 54.71 | 23.05 | 738 | 27,241 | 15 |
| Penn Chinese Treebank[12] | 15,168 | 439,087 | 91.24 | 28.95 | 505 | 31,640 | 31 |
| 東工大コーパス（RWC）[9] | 11,976 | 239,191 | 97.48 | 19.97 | 591 | 22,371 | 31 |
| 東工大コーパス（EDR）[11] | 8,912 | 178,340 | 98.49 | 20.01 | 878 | 19,978 | 31 |

表 1: 実験に用いたコーパス

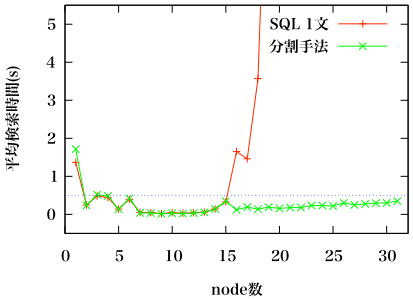


図 4: Penn Treebank Corpus

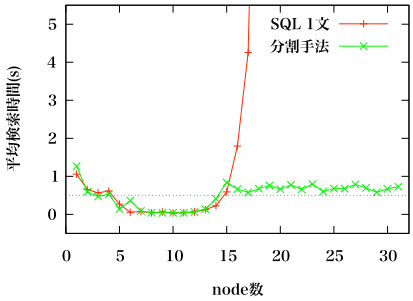


図 5: TIGER Corpora

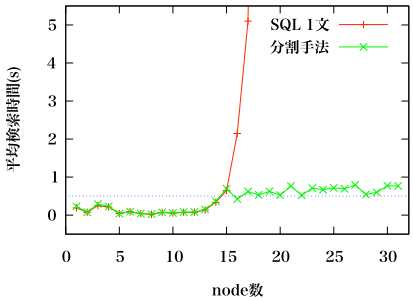


図 6: Penn Korean Treebank

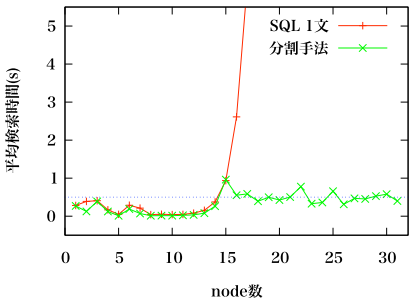


図 7: FLORESTA sintá(c)tica

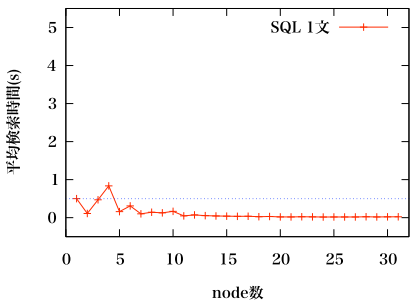


図 8: Penn Chinese Treebank

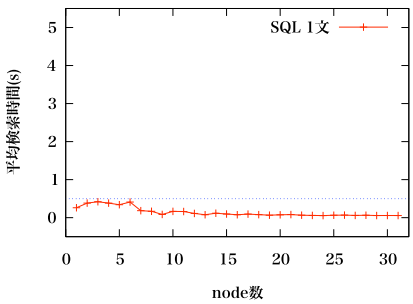


図 9: 東工大コーパス（RWC）

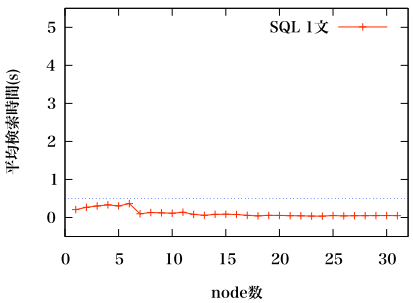


図 10: 東工大コーパス（EDR）

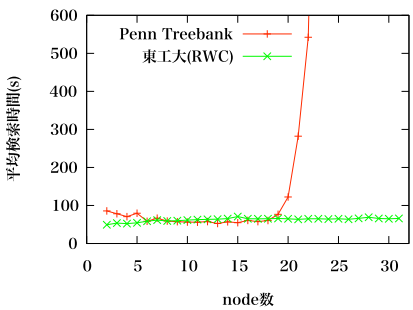


図 11: ラベルなし検索実験

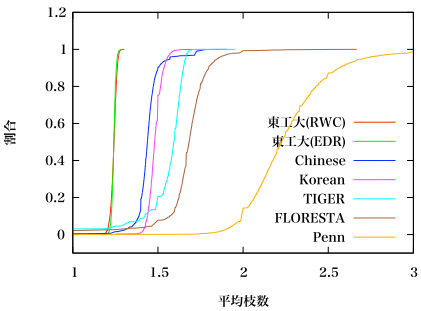


図 12: 平均枝数の分布図