関係データベースを用いた構文木付きコーパス検索手法

橋本　泰一[†]　吉田　恭介[†]　野口　正樹[†]
徳永　健伸[†]　田中　穂積[††]

本論文では，構文木をクエリとして与え，構文木付きコーパスからクエリと同じ構文木を部分木として含む文を検索する手法を提案する．構文木付きコーパスは，関係データベースに格納する．このような構造検索の過去の研究では，クエリの節点数が増加すると，検索時間が大幅に増加する問題があった．本論文で提案する手法は，節点数が多いクエリを部分木に分割し，漸進的に検索することで検索を効率化する．クエリの分割の単位やその検索順序は，検索対象となるコーパス中の規則の出現頻度をもとに自動的に決定する．本手法の有効性を確認するために7種類のコーパスを用いて評価実験を行ったところ，4種類のコーパスで分割の有効性が確認できた．

キーワード:　構文木付きコーパス，情報検索，コーパス作成支援ツール，関係データベース

Retrieving Syntactically Annotated Corpora using a Relational Database

Taiichi Hashimoto[†] , Kyosuke Yoshida[†] , Masaki Noguchi[†] ,
Takenobu Tokunaga[†] and Hozumi Tanaka[††]

This paper presents a method to retrieve sentences including the same subtree as a given query from a treebank. Our system stores the treebank in a relational database. One of the problems of the previous work in structure retrieval is efficiency for large queries. The proposed method divides a large query into several subtrees, and incrementally narrows down the result by using these subtrees as queries. The number of subtrees and the order are determined automaticaly based on the treebank statistics. We conducted experiments to evaluate the proposed method with seven treebanks and found that the proposed method significantly improved the retrieval efficiency in four out of seven treebanks.

KeyWords: *Syntactically Annotated Corpora, Information Retrieval, Annotation Tool, Relational Database*

1　はじめに

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研究の中心となっている．特に，構文木付きコーパスは，統計的手法に基づく言語処理の高性能化のためだけでなく，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

† 東京工業大学 大学院情報理工学研究科 計算工学専攻, Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†† 中京大学 情報科学部 認知科学科, School of Computer and Cognitive Science, Chukyo University

な構文木付きコーパスの作成が必要となっている．しかし，大規模な構文木付きコーパスを全て人手により作成することは，多大なコストを必要とするため困難である．一方，現在の構文解析の精度では，構文木の付与を完全に自動化することが難しい．現実的には，構文解析器の出力から人手によって正しい構文木を選択し，それを文に付与することが望ましい．

コーパス作成中には，文法や品詞体系の変更など，コーパス作成方針の変更により，コーパスへの修正が必要になることもあり，継続的な修正作業や不整合の除去などの機能を持った構文木付きコーパスの作成を支援するシステムが必要になる (Cunningham, Tablan, Bontcheva, and Dimitrov 2003)(Plaehn and Brants 2000)．このようなシステムの多くは，GUI ツールを用いて，構文木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ことにより，コーパス作成者を支援するのが主な機能である．しかし，それだけでは，正しい構文木付きコーパスの作成には，不十分であり，構文木の一貫性を保つための支援が必要となる．

構文木の一貫性を保つための支援として，過去の事例を参照することは有効である．複数の構文木候補のうち，正しい木の選択を迷った場合に，すでに構文木を付与されたコーパス中から，作業中の構文木と類似した部分を持つ構文木を参照できれば，正しい構文木付けが容易になり，一貫性を保つための支援ができる．このためには，構文木付きコーパスを検索対象とし，木構造の検索が可能な構文木付きコーパス検索システムが必要となる．構文木付きコーパス検索システムは，木構造検索を行うことになるため，UNIX の文字列検索コマンド *grep* などの文字列検索よりも検索に時間を要することが多い．既存の構文木付きコーパス検索システム (Randall 2000; Rohde 2001; König, Esther, Lezius, Wolfgang, and an d Holger 2003b; Bird, Chen, Davidson, Lee, and Zheng 2004) においても，主な課題として，検索時間の高速化が挙げられているが，検索時間を高速化する優れた手法はまだ提案されていない．今後，コーパスの規模が更に大きくなると，検索時間の高速化は不可欠な技術となる．

本論文では，高速な構文木付きコーパス検索手法を提案する．本論文で提案する検索手法は，構文木付きコーパスを関係データベースに格納し，検索には SQL を用いる．部分木を検索のクエリとして与え，クエリと同じ構造を含む構文木を検索結果として出力する．クエリの節点数が多い場合，クエリを分割し，それぞれのクエリを別の SQL 文で漸進的に検索する．クエリを分割すべきかどうか，分割するクエリの大きさや検索順序は，構文木付きコーパス中の規則の出現頻度を用いて自動的に決定する．6 言語，7 種類のコーパスを用いて評価実験を行い，4 種類のコーパスにおいて，漸進的に検索を行う本手法により検索時間が短縮され，本手法の有効性を確認した．また，残りの 3 種類のコーパスにおいては，漸進的に検索を行わなくても多大な検索時間を要しないことを本手法で判定することができた．そして，クエリの分割が検索時間の短縮に効果があった 4 種類のコーパスと分割の効果がなかった 3 種類のコーパスの違いについて，コーパスに含まれる文数，ラベルの頻度，節点の平均分岐数の観点から考察を行い，節点の平均分岐数がその一因であることを確認した．

2　構文木付きコーパスのデータベース化

構文木をデータベース化する手法として，XML 文書を関係データベースを用いてデータベース化する吉川らの手法を適用した (吉川正俊, 志村壮是, 植村俊亮 1999)．吉川らの手法は，クエリとして XPath を用い，XML 文書を関係データベースに格納する．XPath とは，W3C により勧告された XML 文書中の特定の構文を表現する記述方法である．XML 文書を検索する場合，クエリである XPath を SQL 文に変換し，クエリを含む XML 文書を検索する．吉川らの手法の大きな特徴は，検索が高速である点である．その理由は，XML 文書の木構造の各節点を出現位置という 2 つの数字で表現し，その大小関係により，節点の親子関係や兄弟関係を表現する点にある．本手法では，構文木の構造を XML 文書の構文構造に対応させ，データベース化を行った．

2.1　出現位置

構文木をデータベースに格納するにあたり，構文木中の各節点対して，出現位置と呼ばれる節点間の関係を計算するための 2 つの数字を与える．出現位置は，(*left_position* ，*right_position*) の対で表現され，次のアルゴリズムにより決定する．

- 葉
  左端から N 番目の葉に対する出現位置は，(N, N) という整数値の対を与える．
- 葉以外の節点
  変数 *position* を 1 以下の微小値 α で初期化する．根から深さ優先探索で辿り，節点を辿るときに α を *position* に加算し，葉を辿るきに葉の *left_position* または *right_position* の値を代入する．節点を最初に通過するとき変数 *position* の値を *left_position* として，最後に通過するときに *right_position* として決定する．ただし，α は，木の最大の深さの逆数よりも小さい値でなければならない．

このアルゴリズムで決定した出現位置の例を図 1に示す．また，このように各節点に出現位置を与えることで，図 2のように節点間の関係を出現位置の大小関係で表現することができる．

2.2　関係データベースへの格納

前節で計算した出現位置とともに各節点の情報を関係データベースに格納する．データベースは，*Node Table*，*Document Table*，*Label Table* の 3 つのテーブルにより構成される．テーブルの例をそれぞれ表 1，2，3に示す．

Node Table は，構文木を構成する各節点に関する情報を格納する．各項目は，*ID* が節点固有の値，*parentID* が節点の親の *ID*，*docID* が節点を含む構文木の *ID*，*labelID* が節点のラベルの *ID*，*nextSibID* が節点の右隣の兄弟の *ID*，*l_pos, r_pos* が節点の出現位置を表す．

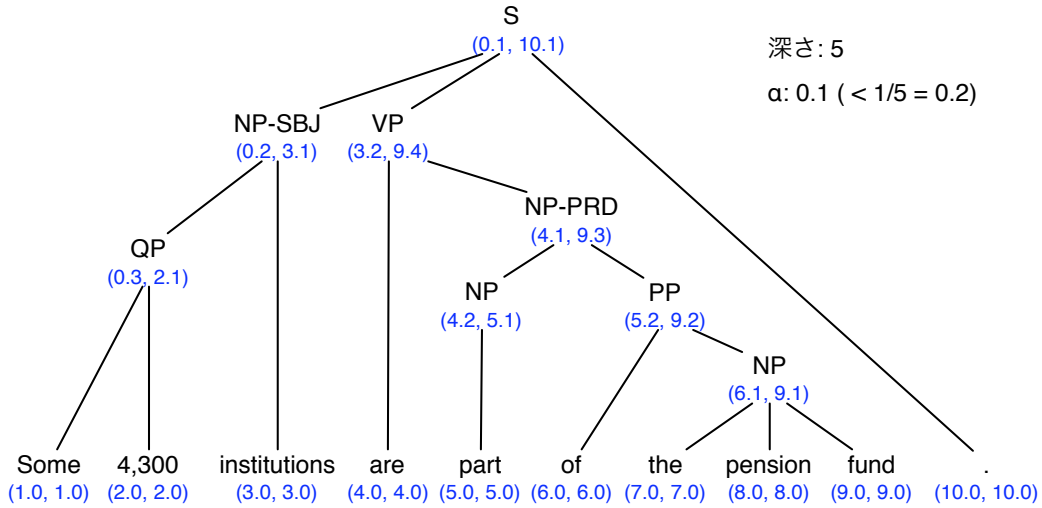


図 1　出現位置の例

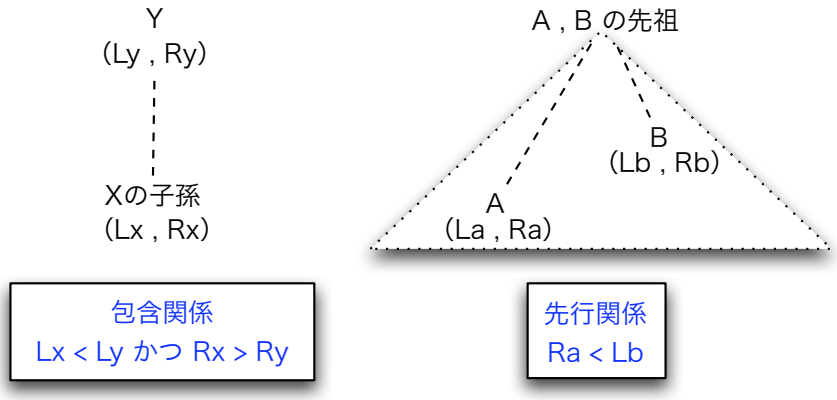


図 2　包含関係と先行関係

*Document Table*は，構文木が記述されているファイルに関する情報を格納する．各項目は，*docID*が構文木固有の値，*file*が構文木が記述されているファイル名を表す．

*Label Table*は，コーパスに含まれる記号，単語に関する情報を格納する．各項目は，*labelID*がラベル固有の値，*label*がラベル名，*frequency*がコーパスにおけるラベルの頻度を表す．

表 1　Node Table

ID	docID	labelID	parentID	nextSibID	l_pos	r_pos
179844	3924	1	0	0	0.1	10.1
179845	3924	2	179844	179850	0.2	3.1
179846	3924	79	179845	179849	0.3	2.1
179847	3924	352	179846	179848	1.0	1.0
179848	3924	12051	179846	0	2.0	2.0
179849	3924	564	179845	0	3.0	3.0
179850	3924	11	179844	179861	3.2	9.4
179851	3924	266	179850	179852	4.0	4.0
179852	3924	27	179850	0	4.1	9.3
179853	3924	3	179852	179855	4.2	5.1
179854	3924	1768	179853	0	5.0	5.0

ID:　節点固有の値
docID:　節点を含む構文木の ID
labelID:　節点のラベルの ID
nextSibID:　節点の右隣の兄弟の ID
parentID:　節点の親の ID
l_pos:　節点の位置情報
r_pos:　節点の位置情報

表 2　Document Table

docID	file
1	wsj_0001.xml
2	wsj_0002.xml

docID:　構文木固有の値
file:　ファイル名

表 3　Label Table

labelID	label	frequency
1	“S”	106,901
2	“NP-SBJ”	94,319
79	“QP”	11,405
352	“Some”	1,936
12051	“4,300”	1
564	“institutions”	155
11	“VP”	179,161
266	“are”	4,508
27	“NP-PRD”	7,566
3	“NP”	290,484

labelID:　ラベル固有の値
label:　ラベル名
frequency:　ラベルの頻度

3　構文木付きコーパスの検索手法

3.1　クエリの定義

前節では，構文木付きコーパスを関係データベースに格納する方法について述べた．本手法では，木構造をクエリとし，クエリを部分木として含む文を検索する．まず，クエリの例を図3に示す．図中の “∗” は，任意のラベルを意味する．さらに，クエリを部分木として含むかどうかの判定方法として，完全一致と部分一致の二種類を用意した．完全一致は，クエリの各節点の分岐数とコーパス内の対応する節点の分岐数が一致しなければならない．一方，部分一致は，

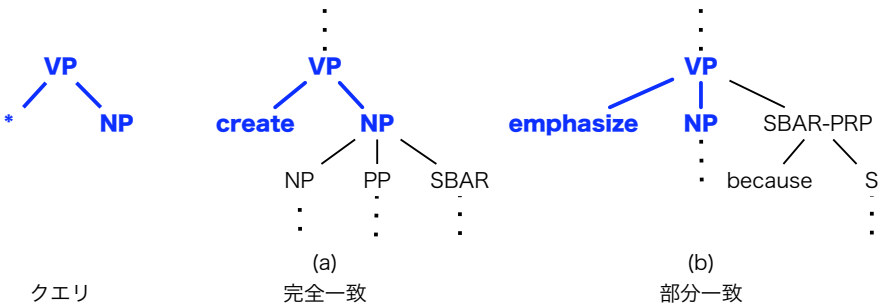


図 3　完全一致，部分一致の例

この分岐数が必ずしも一致する必要がない．例えば，図 3において，完全一致で検索した場合，(a) の木は，各節点の分岐数，ラベルが一致するためにクエリと一致すると見なす．一方，(b) の木は，分岐数が異なるため，クエリと一致すると見なさない．しかし，部分一致で検索した場合，節点の分岐数は一致する必要がないため，両方の木がクエリと一致すると見なす．

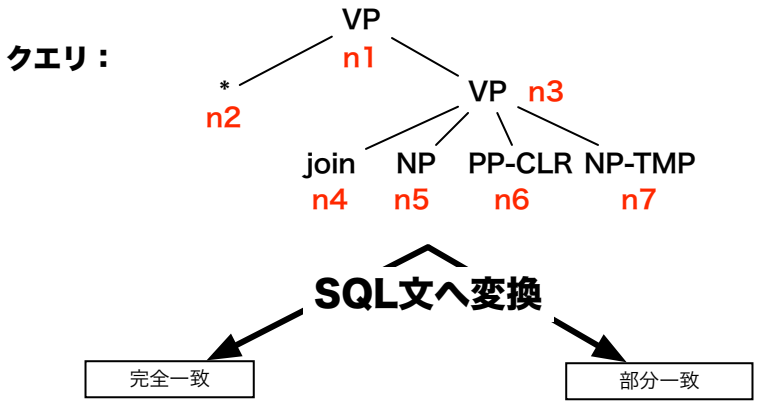
3.2　構文木をクエリとした検索手法

本手法は，吉川らの手法と同様に検索を行うためにクエリである部分木を関係データベースのデータの操作，定義，検索などを行う言語SQLの文へ変換する．そして，変換したSQL文により該当した構文木をデータベース内から検索し，クエリを部分木として含む構文木を得る．クエリとそのクエリに対応するSQL 文を図 4に示す．図 4中のSQL文の where構文以降が各節点の条件式となっている．システムは，条件を満たす節点をデータベース内を検索し，すべての条件を満たす節点をもつ構文木を出力する．

3.3　予備実験

吉川らは，シュークスピアの戯曲を Jon Bosak がタグ付けしたXML文書 (Bosak 1999) を用いた評価実験により，検索手法が高速であることを示した (吉川正俊他 1999)．そこで，吉川の手法を構文木付きコーパスに適用し，予備実験を行った．検索対象のコーパスとして，Penn Treebank Corpus（48,884文）(Marcus, Kim, Marcinkiewicz, MacIntyre, Bies, Ferguson, Katz, and Schasberger 1994) を用いた．クエリは，Penn Treebank Corpus からランダムに4文を抽出した．そして，4文から節点数が2 から 20 の抽出可能なすべての部分木をクエリとして，部分一致により検索を行った．その結果を表 4，図 5に示す．

実験結果より，クエリの節点数が7から12の間は，高速に検索を行っているが，節点数が7以下，12以上の場合，検索時間が非常に増加している．クエリの節点数が7以下の場合，クエ




```
select
    n1.docID
from
    node n1, node n2,
    node n3, node n4,
    node n5, node n6,
    node n7
where
    n2.parentID = n1.ID
    and n2.nextSibID = n3.ID
    and floor(n1.l_pos) = floor(n2.l_pos)
    and n3.nameID = 11
    and n3.parentID = n1.ID
    and n3.nextSibID = 0
    and n4.nameID = 13
    and n4.parentID = n3.ID
    and n4.nextSibID = n5.ID
    and floor(n3.l_pos) = floor(n4.l_pos)
    and n5.nameID = 3
    and n5.parentID = n3.ID
    and n5.nextSibID = n6.ID
    and n6.nameID = 16
    and n6.parentID = n3.ID
    and n6.nextSibID = n7.ID
    and n7.nameID = 21
    and n7.parentID = n3.ID
    and n7.nextSibID = 0
    and n1.nameID = 11
```

```
select
    n1.docID
from
    node n1, node n2,
    node n3, node n4,
    node n5, node n6,
    node n7
where
    n2.parentID = n1.ID
    and floor(n2.r_pos) <= floor(n3.l_pos)
    and n3.nameID = 11
    and n3.parentID = n1.ID
    and n4.nameID = 13
    and n4.parentID = n3.ID
    and floor(n4.r_pos) <= floor(n5.l_pos)
    and n5.nameID = 3
    and n5.parentID = n3.ID
    and floor(n5.r_pos) <= floor(n6.l_pos)
    and n6.nameID = 16
    and n6.parentID = n3.ID
    and floor(n6.r_pos) <= floor(n7.l_pos)
    and n7.nameID = 21
    and n7.parentID = n3.ID
    and n1.nameID = 11
```

図 4　クエリとそのSQL文の例

表 4　予備実験結果

節点数	平均検索時間（秒）	最大検索時間	最小検索時間	平均出力数（部分木）	クエリ数
2	2.04	10.63	0.001	24330.30	65
3	1.59	11.33	0.001	9854.96	99
4	0.84	11.51	0.002	2189.00	168
5	0.38	10.05	0.002	337.26	281
6	0.19	6.42	0.002	65.11	437
7	0.12	1.64	0.003	17.00	611
8	0.09	1.64	0.003	7.16	762
9	0.07	1.35	0.004	3.32	860
10	0.06	1.31	0.008	1.91	902
11	0.05	1.31	0.009	1.36	901
12	0.07	1.32	0.010	1.13	869
13	0.11	3.84	0.012	1.04	801
14	0.18	9.36	0.018	1.01	678
15	0.28	45.45	0.041	1.00	498
16	0.56	2.25	0.065	1.00	300
17	1.94	6.12	0.270	1.00	140
18	7.51	18.09	1.093	1.00	47
19	31.13	47.52	8.021	1.00	10
20	161.02	185.45	135.25	1.00	3

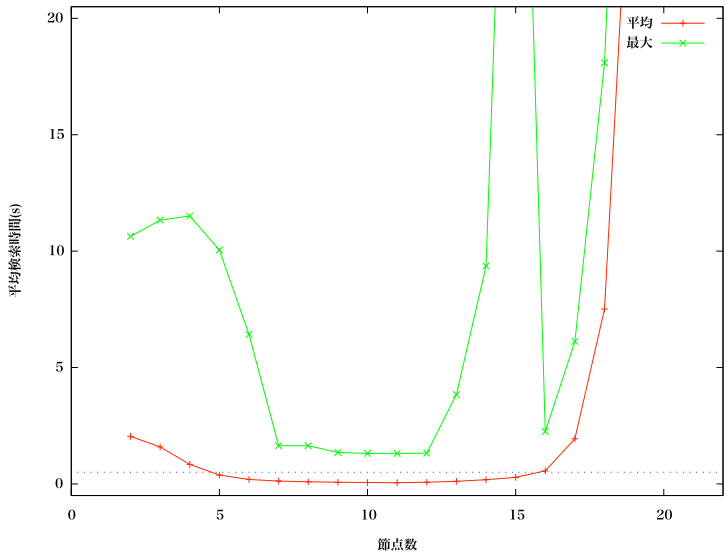


図 5　予備実験結果

リに一致する文が非常に多いために検索に時間を要とすると考えられる．一方，クエリの節点数が12以上の場合は，クエリと一致する文かどうかの判定，つまりSQL文の条件判定に検索時間を要していると考えられる．検索条件が多ければ，条件判定を行う処理が多くなり検索に時間を要する．吉川らが評価実験を行ったXML文書は，DTD（Document Type Definition）により，節点のラベルや構造があらかじめ定義されている．構文木付きコーパスにおいては，文脈自由文法がDTDに相当する．XML文書は，少ないDTDの規則で簡潔な構造を定義している場合が多く，文書間での大きな構造の違いは少ない．そのため，SQL文の条件は多くなりにくい．一方，構文木付きコーパスは，数千の規則を用いて文法を定義し，文によって様々な木構造が付与されている．そのため，XML文書よりもSQL文の条件が多くなる可能性が高く，検索時間が増加するケースが多くなることが予想できる．

関係データベースの検索速度は，関係データベースシステムの種類，格納するデータ，検索に要するSQL文に依存する．関係データベースを構築する際，技術者が格納するデータをみて関係データベースシステムを選んだり，生成するSQL文の条件部分の優先度などの調整を経験的に行い，検索の効率化をはかることが多い．クエリの節点数が多くなると変換したSQL文の条件部分が多くなり，条件を満たすデータの検出時間がかかる．そのため，検索時間が大幅にかかるようになる．このような場合，人手により条件の記述順序の変更などにより，チューニングを行う．しかし，構文付きコーパスは，言語や対象文書の違いなど様々なコーパスが存在する．コーパスそれぞれを人手でチューニングすることは困難である．そのため，自動的に検索速度をチューニングする手法が必要である．

3.4　漸進的検索

予備実験の結果から，クエリの節点数が7から12の間は，高速に検索が可能であることがわかった．例えば，節点数16のクエリの平均検索時間は，0.56 秒である．それに対して，節点数8のクエリの平均時間は，0.09秒である．もし，節点数16のクエリを節点数8のクエリ2つに分割し検索できれば，検索を高速化することが可能である．

本手法では，クエリをSQL一文で検索するのではなく，複数のSQL文に分割し，SQL一文に要する検索時間を短くすることで高速化を行う．また，単一のSQL文により効率よく絞り込みが行うことができるように，コーパス内のラベルの頻度をもとにクエリの分割方法を決定する．

検索単位

クエリの節点数が多い場合には，クエリを複数のクエリに分割し，漸進的に検索を行う手法を提案する．本論文では，分割されたクエリを検索単位，効率的に検索可能なクエリの節点数の最大値を最大検索単位節点数と定義する．コーパスが与えられたとき，コーパスから最大検

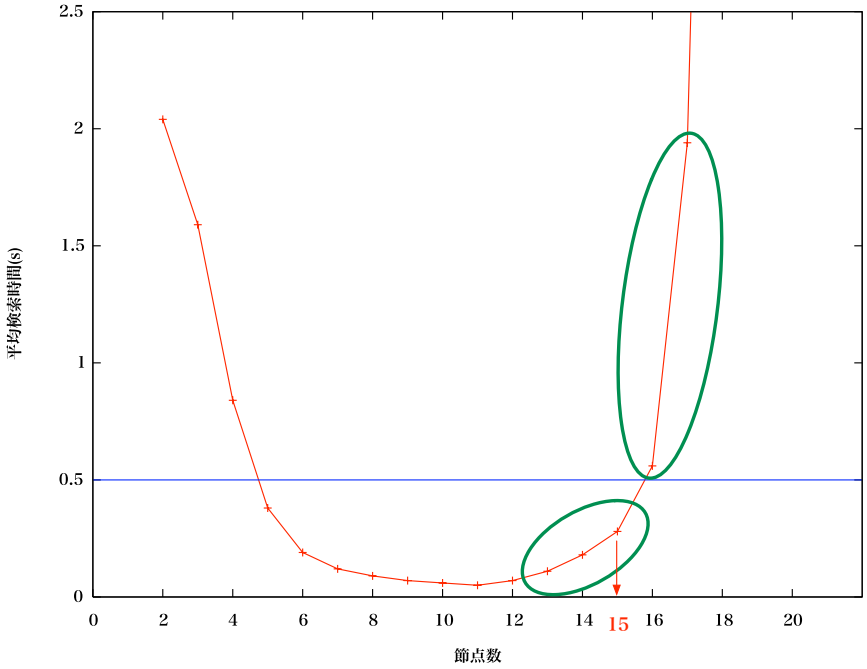


図 6　最大検索単位節点数決定例

索単位節点数を計算し，ノード数が多いクエリは最大検索単位節点数をもとに分割して，漸進的に検索を行う．

まず，コーパスから，節点数2から31[1]のクエリ（部分木）各50個をランダムに抽出する．そして，各節点数ごとに平均検索時間を算出する．そのデータから，以下の二つの条件を満たす最大値iを最大検索単位ノード数とする．

- 節点数 $i-n+1$ から i の間のクエリの平均検索時間がt以下
- 節点数 $i+1$ から $i+n$ の間のクエリの平均検索時間がt以上

nとtは，最大検索単位節点数を決定する際のパラメータである．nは，正の整数，tは，秒数である．例えば，nを3，tを0.5（秒）をした場合，図 6の予備実験結果では，節点数13，14，15において平均検索時間がtを下回り，節点数16，17，18において平均検索時間がtを上回っている．このとき，最大検索単位節点数は，15となる．

1 節点数が31までなのは，実験で用いた関係データベースシステムのSQL文は，クエリとして最大31ノードまでしか制約をかけることができないためである

クエリの分割

前節で，コーパスに最適化された検索単位の節点数を決定する手法について述べた．しかし，クエリの分割方法，分割されたクエリの検索順序によって，検索時間は大きく変わる．もし，最初に検索する検索単位の出力数が少なければ，次に絞り込む検索範囲が狭まり，検索を効率よく行うことができる．つまり，絞り込みが早く行われるように分割や順序を決定することが望ましい．コーパス中の節点や文脈自由規則の頻度をもとにクエリの検索単位への分割，検索単位の検索順序を決定する．クエリの分割アルゴリズムを以下に示す．検索単位の検索順序は，分割された順である．

Step 1　クエリに含まれる節点のラベルのコーパス内での出現頻度を計算する．

Step 2　クエリ内の最小頻度の節点を検索単位U_0とする．

Step 3　$i = 1$

Step 3　検索単位U_{i-1}に含まれる節点に近接する最小頻度のラベルの節点で初期化する．

Step 4　検索単位U_iに含まれる節点を持ち，検索単位U_iに加えても最大検索単位節点数を越えない部分木があれば，根の頻度が最小である部分木の節点を検索単位U_iへ追加する．

Step 5　もし，Step 4 において，部分木を加えられたのであれば，Step 4へ．そうでなければ，Step 6へ．

Step 6　$i = i + 1$．クエリをすべて分割したならば，Step 7へ．そうでなければ，Step 3へ．

Step 7　各検索単位をSQL文へ変換．

クエリ分割方法を例を用いて説明する．クエリとして，図 7の（1）が与えられ，最大検索単位節点数が5 であると仮定する．まず，クエリに含まれる節点のラベルの出現頻度を計算する．その中で，最も出現頻度が低い “join”（出現頻度50）を検索単位U_0に加える．次に，節点（“join”）を含む部分木の中で，最も出現頻度が低い “VP” を根とする部分木（出現頻度179,161）の節点を加えることを考える．しかし，この部分木の節点を加えると最大検索単位節点数を越えるためU_0には加えない．（図 7の（2））次に，新たな検索単位U_1をU_0に含まれる節点と隣接する節点のうち最小頻度である節点（“VP”）で初期化する．そして，“VP” を含む部分木をU_1に加えることを考える．“VP” を根とする部分木の節点を加えた場合，U_1の節点は，“VP”, “NP”, “PP-CLR”, “NP-TMP” の4つとなり，最大検索単位節点数を越えない．そのため，これらの節点をU_1に追加する．さらに，U_1に “VP” を根とする部分木の節点を加えることを試みるが最大検索単位節点数を越えるために加えられない．（図 7の（3））そして，新たな検索単位U_2を構築を始める．

最終的に，図 7の（4）のように，クエリは3つの検索単位U_0，U_1, U_2に分割される．そして，それぞれの検索単位をSQL文へ変換し，検索を行う．そして，最初に決定した検索単位U_0により検索を行い，その結果に対して検索単位U_1により絞り込みを行う．絞り込みは，検索単

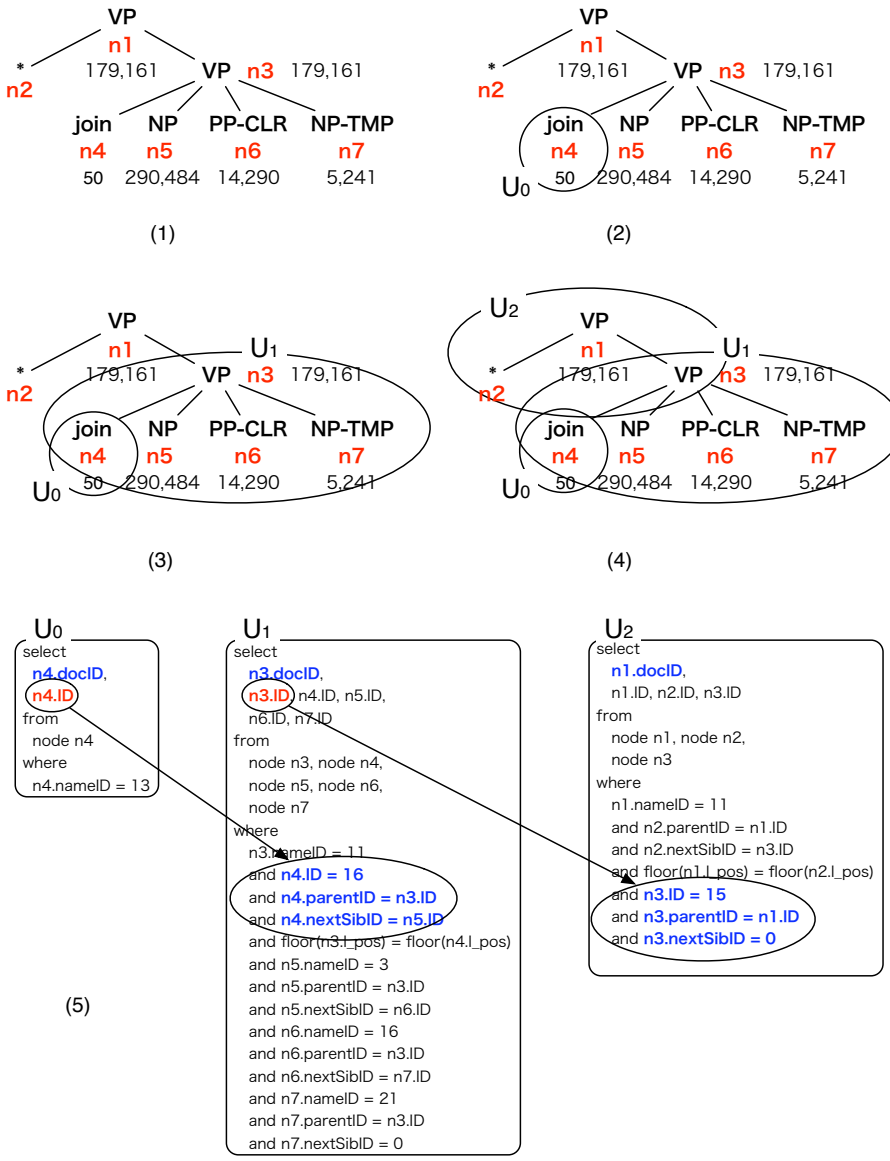


図 7　クエリ分割の例

位の初期化に利用した隣接する節点を次の検索単位のSQL文内の条件式として追加する．つまり，直前に利用したSQL文で次の検索単位に隣接する節点のIDを獲得し，その条件を次の検索単位のSQL文の条件として加える．例では，U_1 に対応するSQL文の節点 “n4” に関する条件式，U_2 に対応するSQL文の節点 “n3” に関する条件式が追加される条件式である．(図 7の（5)）

表 5　評価実験に用いたコーパス

コーパス名	文数	単語数 (異なり)	1文当りの 平均節点数 (単語数)	非終端記号 異なり数	最大検索単位節点数 完全一致/部分一致
Penn Treebank	48,884	1,244,104 (44,175)	45.34 (25.45)	1,225	15/11
TIGER	40,020	616,414 (73,385)	39.24 (15.40)	723	15/12
Penn Korean Treebank	5,083	102,095 (3,317)	59.48 (20.09)	123	15/13
FLORESTA sint*á*(c)tica	8,31	191,476 (27,241)	54.71 (23.05)	738	15/12
Penn Chinese Treebank	15,168	439,087 (31,640)	91.24 (28.95)	505	31/31
東工大（RWC）	11,976	239,191 (22,371)	97.48 (19.97)	591	31/31
東工大（EDR）	8,912	178,340 (19,978)	98.49 (20.01)	878	31/31

4　評価実験

クエリ分割による検索の有効性を確認するために，7つの構文木付きコーパスを用いて，評価実験を行った．評価実験に使用したコーパスは，Penn Treebank Corpus(Marcus et al. 1994)，TIGER Corpus(K*ö*nig, Esther, Lezius, and Wolfgang 2003a)，Penn Korean Treebank(hye Han, Han, Ko, Yi, and Palmer 2002)，FLORESTA sint*á*(c)tica(Afonso, Bick, Haber, and Santos 2002)，Penn Chinese Treebank(Xue, Chiou, and Palmer 2002)，東工大コーパス（RWC）(Noro, Hashimoto, Tokunaga, and Tanaka 2004)，東工大コーパス（EDR）(野呂智哉, 白井清昭, 徳永健伸, 田中穂積 2002) である．各コーパスの諸元を表 5に示す．

評価実験では，最大検索単位節点数を決定するパラメータtを0.5（秒），nを3とし，最大検索単位節点数を計算した．クエリは，コーパスからランダムに抽出した部分木を用い，節点数1から31までの各50個を用いた．検索方法としては，完全一致と部分一致の2種類を行った．各コーパスにおける評価実験結果を図 8から図 21に示す．

Penn Treebank Corpus，TIGER Corpus，Penn Korean Treebank Corpus，FLOREST sint*á*(c)ticaの4つコーパスにおいては，完全一致による検索では節点数16以上，部分一致による検索では節点数11から13以上のクエリに対して，クエリの分割を行わずに検索を行った場合，急激に検索時間が増大することがわかる．一方，クエリを検索単位に分割することで高速に検索が行うことができた．また，対照的に Penn Chinese Treebank Corpus，東工大コーパス（RWC），東工大コーパス（EDR）では，完全一致と部分一致ともに許容検索時間を越えることがなく，最大検索単位節点数は最大の31と計算された．これら三つのコーパスにおいては，

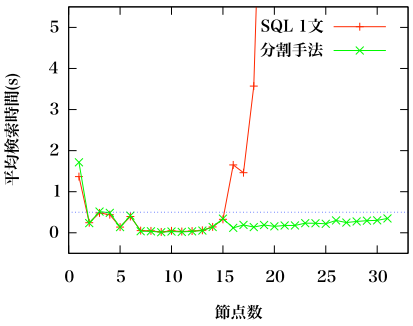


図 8　Penn Treebank Corpus: 完全一致

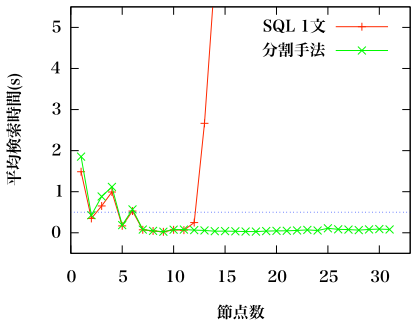


図 9　Penn Treebank Corpus: 部分一致

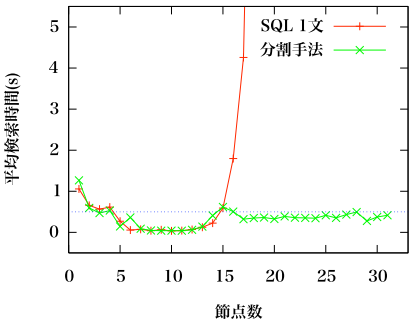


図 10　TIGER Corpus: 完全一致

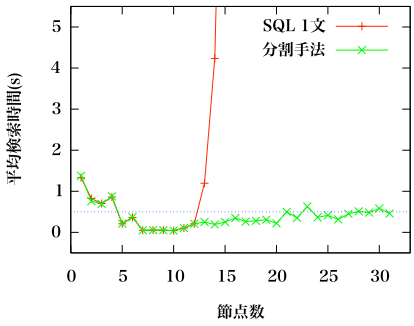


図 11　TIGER Corpus: 部分一致

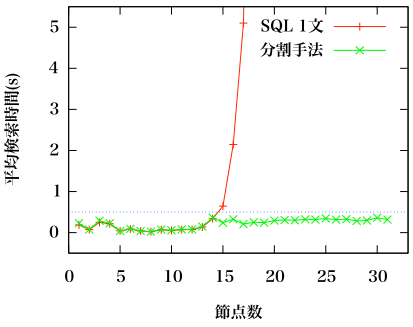


図 12　Penn Korean Treebank: 完全一致

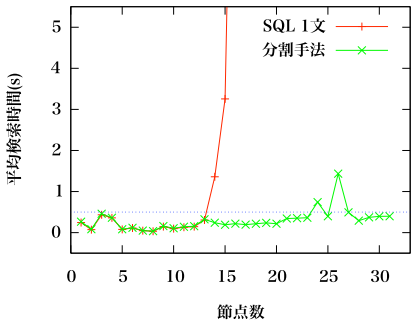


図 13　Penn Korean Treebank: 部分一致

クエリの分割を行う必要がなかった.

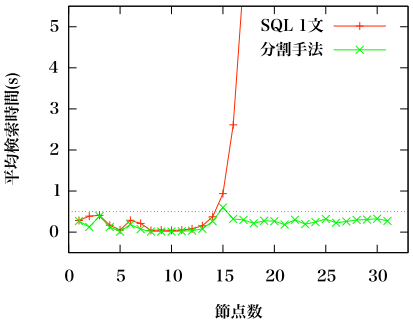


図 14　FLORESTA sintá(c)tica: 完全一致

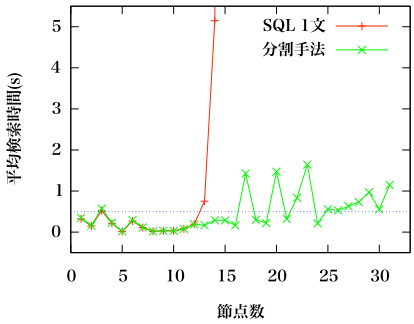


図 15　FLORESTA sintá(c)tica: 部分一致

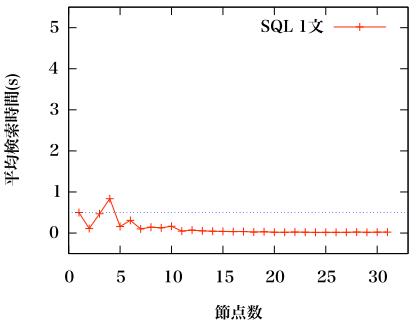


図 16　Penn Chinese Treebank: 完全一致

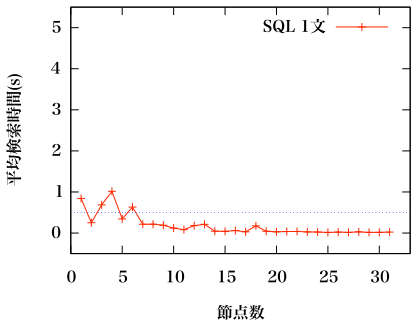


図 17　Penn Chinese Treebank: 部分一致

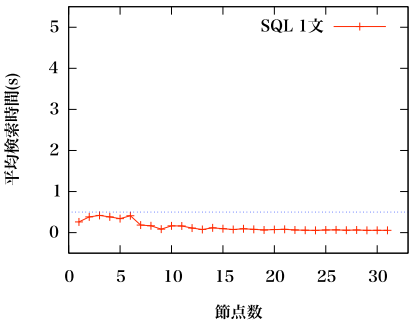


図 18　東工大コーパス（RWC）：完全一致

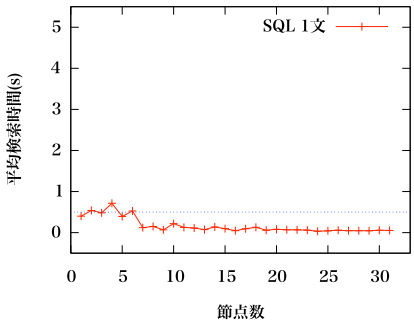


図 19　東工大コーパス（RWC）：部分一致

5　考察

評価実験で用いたコーパスは，大きく分けて二つに分類できる．一つは，分割せずに検索した場合，クエリを構成する節点数が増えると検索時間が非常に遅くなるものである．このタイプのコーパスは，Penn Treebank Corpus，TIGER Corpus，Penn Korean Treebank Corpus,

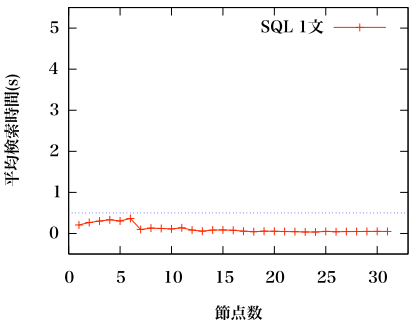


図 20　東工大コーパス（EDR）：完全一致

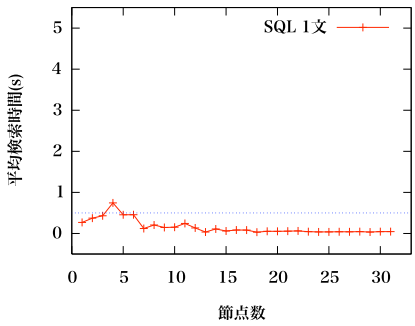


図 21　東工大コーパス（EDR）：部分一致

FLOREST sintá(c)ticaが当てはまる．もう一つは，クエリの節点数が増えても検索時間に影響を与えないものである．このタイプのコーパスとして，Penn Chinese Treebank Corpus，東工大コーパス（RWC），東工大コーパス（EDR）が当てはまる．このような違いが，コーパスのどの特徴と関連しているのか，コーパスの以下の特徴に着目し考察した．

- コーパスの文数
- ラベルの頻度
- 節点の平均分岐数

5.1　コーパスの文数

コーパスの文数が検索時間に影響を与えることが考えられる．つまり，文数が多ければ検索に時間がかかり，少なければ時間がかからないはずである．しかし，表 5からわかるようにPenn Korean Treebank Corpus（5,083文），FLOREST sintá(c)tica（8,307文）は，比較的コーパスに含まれる文数が少ないが，クエリの節点数が増加するとともに検索時間が大幅に増加している．また，Penn Chinese Treebank（15,168文）は文数が多いにも関わらず，節点数が増加しても検索時間は増加していない．

さらに Penn Treebank Corpus の文数を1,000文，5,000文，11,976文，48,884文と変化させ，検索時間の変化を調べた．クエリなどの実験環境は，評価実験で用いたものと同じである．その結果を図 22に示す．この結果からコーパスに含まれる文数が減少しても，節点数が増加するとともに検索時間が増大していることがわかった．つまり，コーパスの文数と検索時間増大の間には，関連性は低いものと考えられる．

5.2　ラベルの頻度

クエリ内のラベルが検索時間に影響を与えることが考えられる．例えば，クエリに含まれるラベルが非常に稀にしか出現しないラベルであれば，非常に早く検索することができるはずで

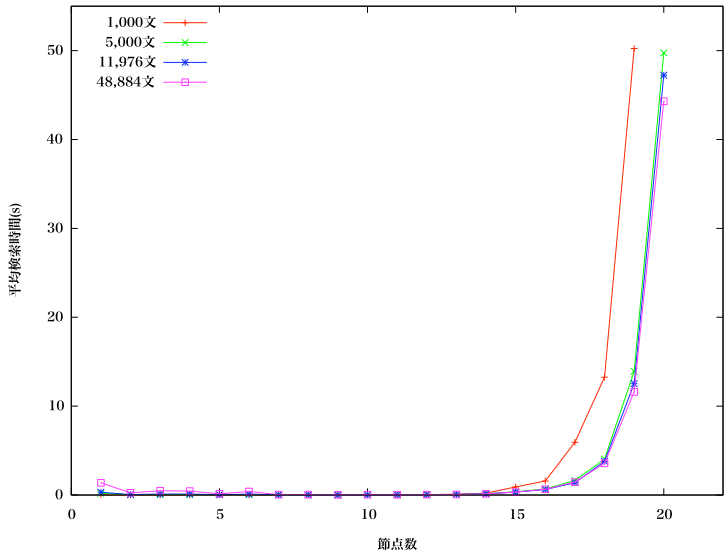


図 22　コーパスの文数と検索時間

ある．しかし，表 5からわかるように，Penn Treebank Corpus はラベル（非終端記号）の異なり数が多いが検索時間が急激に増加している．一方，東工大コーパス（RWC）はラベルの異なりが少ないが，検索時間が増大していない．

Penn Treebank Corpus，東工大コーパス（RWC）を用いて，ラベルを “∗” で置き換えたクエリを用いて，実験を行った．クエリは，評価実験を行ったときと同一である．その結果を図 23に示す．ラベルの情報がなくなったため，双方ともに評価実験よりも検索時間を要しているが，節点数と検索時間との関係に変化はなかった．ラベルの頻度と検索時間の増大は関連性が低いと考えられる．

5.3　節点の平均分岐数

次に，コーパスの特徴として節点の平均分岐数に注目した．分岐数が多くなれば，構文木は複雑になり，検索に時間を要することが考えられる．そこで，全てのコーパスについて，それぞれの節点の平均分岐数を調べた．その結果を図 24に示す．この結果から確かに検索時間が増大する傾向にあるコーパスは，平均子供数が比較的大きいことがわかる．しかし，Penn Korean Treebank と Penn Chinese Treebank の差はあまりなく，平均分岐数が決定的な原因であるとは考えがたい．だが，コーパス内の文の構造の複雑さが一因である可能性は確認できる．

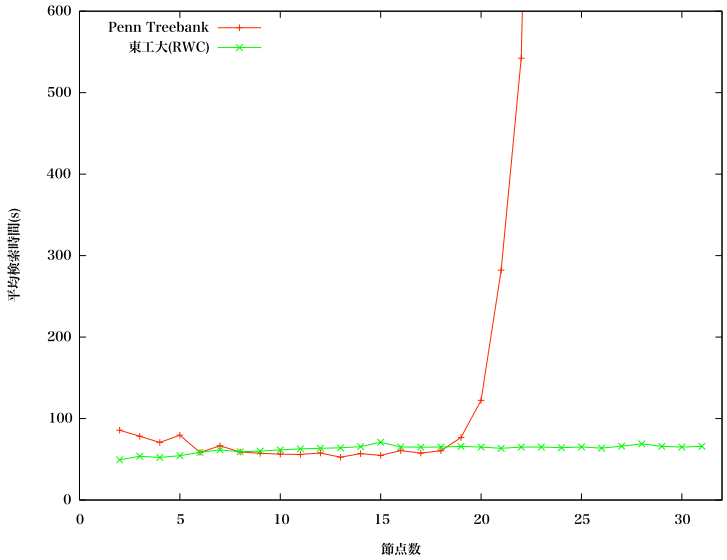


図 23　ラベルなしクエリによる実験結果

6　まとめ

本論文では，構文木付きコーパスを高速に構造検索する手法を提案した．構文木付きコーパスのデータベース化は，木構造を持つXML文書を格納・検索する手法を用い，部分木で表現されたクエリをSQL文に変換して検索を行う．節点数の多いクエリに対する検索時間が増大するという問題を解決するために，クエリを分割し，漸進的に検索する手法を提案した．クエリの分割は，まずコーパスから最大検索単位節点数を計算し，最大検索単位節点数をもとに，効率的に検索できるようクエリを分割し，検索を行う．

評価実験では，7つのコーパス中の4つに対して，クエリを分割せずに検索する手法よりも効率的に検索を行うことができることを示した．また，評価実験で用いたコーパスには，クエリの節点数が増加すると検索時間が非常に遅くなるものと，そうでないものの2種類が存在した．その違いの要因が，コーパスに含まれる構文構造の複雑さに起因するものであることについて考察した．

今後の課題として，次のようなものがあげられる．

- 節点が少ないクエリに対する検索の高速化
- 検索時間とコーパスに含まれる構文構造の複雑さとの関連性の解明
- コーパス作成支援システムへの応用

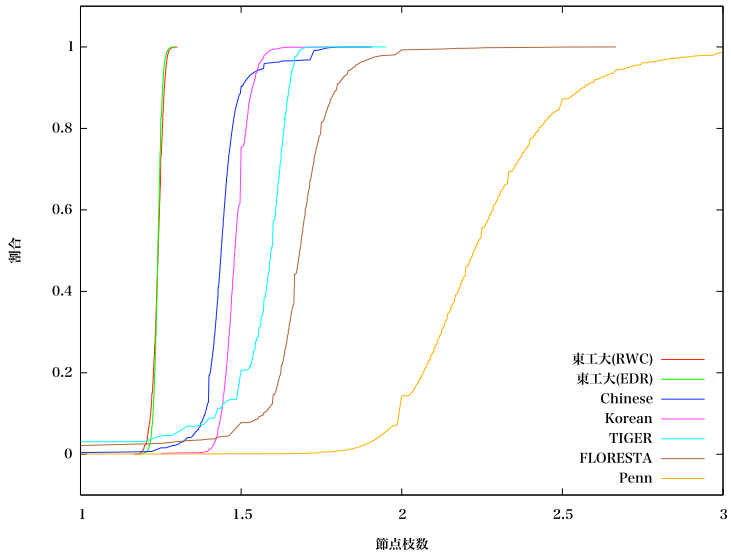


図 24　節点の平均分岐数

参考文献

Afonso, S., Bick, E., Haber, R., and Santos, D. (2002). ""Floresta sintá(c)tica": a treebank for Portuguese." In *Proceedings of LREC 2002*, pp. 1698–1703.

Bird, S., Chen, Y., Davidson, S., Lee, H., and Zheng, Y. (2004). "Extending XPath to Support Linguistic Queries." In *UWA Language Science Group 2004 Program - Symposium on Speech a nd Language Technology*.

Bosak, J. (1999). *The Plays of Shakespeare in XML*.

Cunningham, H., Tablan, V., Bontcheva, K., and Dimitrov, M. (2003). "Language Engineering Tools for Collaborative Corpus Annotation." In *Proceedings of Corpus Linguistics 2003*.

hye Han, C., Han, N.-R., Ko, E.-S., Yi, H., and Palmer, M. (2002). "Penn Korean Treebank: Development and Evaluation." In *Proceedings of the 16th Pacific Asia Conference on Language, Inform Ation and Computation*.

König, Esther, Lezius, and Wolfgang (2003a). "The TIGER Language - A Description Language for Syntax Graphs, Formal Definition." Tech. rep., Institut für Maschinelle Sprachverarbeitung, University of Stuttgart.

König, Esther, Lezius, Wolfgang, and an d Holger, V. (2003b). *TIGERSearch User's Manual*.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz,

K., and Schasberger, B. (1994). “The Penn Treebank: Annotating Predicate Argument Structure.” In *ARPA ’94*.

Noro, T., Hashimoto, T., Tokunaga, T., and Tanaka, H. (2004). “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*, pp. 115–126.

Plaehn, O. and Brants, T. (2000). “Annotate – An Efficient Interactive Annotation Tool.” In *Proceedings of the 6th Conference on Applied Natural Language Processing*.

Randall, B. (2000). *CorpusSearch User’s Manual*.

Rohde, D. (2001). *Tgrep2 User Manual*.

Xue, N., Chiou, F.-D., and Palmer, M. (2002). “Building a Large-Scale Annotated Chinese Corpus.” In *Proceedings of COLING 2002*.

吉川正俊, 志村壮是, 植村俊亮 (1999). “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, **40** (SIG6(TOD3)), 115–131.

野呂智哉, 白井清昭, 徳永健伸, 田中穂積 (2002). “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.

略歴

橋本 泰一 (正会員): 1997年東京工業大学工学部情報工学科 卒業．2002年同大学大学院情報理工学研究科博士課程修了．同年同大学大学院情報理工学研究科 助手．現在，同大学統合研究院 特任准教授．博士（工学）．自然言語処理，知識情報処理の研究に従事．言語処理学会，情報処理学会各会員．

吉田 恭介: 2003年東京工業大学工学部情報工学科卒業．2005年同大学大学院情報理工学研究科修士課程修了．

野口 正樹: 2004年東京工業大学工学部情報工学科卒業．2006年同大学大学院情報理工学研究科計算工学専攻修士課程修了．現在，同専攻博士課程に在学中．言語資源の検索・利用に関する研究を行っている．また，言語資源の構築や知的創造作業支援などにも興味を持っている．情報処理学会学生会員

徳永 健伸 (正会員): 1961年生．1983年東京工業大学工学部情報工学科卒業．1985年同大学院理工学研究科修士課程修了．同年(株)三菱総合研究所入社．1986年東京工業大学大学院博士課程入学．現在，同大学大学院情報理工学研究科准教授．博士 (工学)．自然言語処理，計算言語学，情報検索などの研究に従事．情報処理学会，人工知能学会，言語処理学会，計量国語学会，Association for Computational Linguistics，ACM SIGIR各会員．

田中 穂積 (正会員): 1964年東京工業大学工学部情報工学科卒業．1966年同大学院理工学研究科修士課程修了．同年電気試験所（現産業技術総合研究所）

入所．1980年東京工業大学助教授．1983年東京工業大学教授．現在，中京大学理工学部教授．工学博士．人工知能，自然言語処理に関する研究に従事．情報処理学会，電子情報通信学会，認知科学会，人工知能学会，言語処理学会，計量国語学会，AAAI，各会員．