リレーショナルデータベースを用いたコーパスからの情報抽出：その方法について[1]

西村 祐一[*]・滝沢 直宏[**]

要旨

コーパスは、多くの場合、テキストファイルで構築されている。したがって、そこからの情報抽出は、特別なコーパス分析用ソフトを用いずとも、テキスト処理ツールのみによって行うことができる。しかし、この方法では、処理過程が透明になるという利点はあるものの、picture のような形式で情報を得るには処理が複雑になり、データ量に応じて処理時間も長くなるという欠点がある。本稿では、リレーショナルデータベースを用いた情報抽出システムの基本的設計を論じ、その解決策を示す。

キーワード： コーパス、テキストファイル、picture、リレーショナルデータベース

* 名古屋大学 大学院国際開発研究科 国際コミュニケーション専攻 博士後期課程 満期退学
** 立命館大学 大学院 言語教育情報研究科 教授

1. テキストファイルとしてのコーパスと処理ツール

コーパス(電子化された言語資料)は、(記述的・理論的)言語研究の推進という明確な目的をもって構築された狭い意味でのコーパスであれ、新聞のデータベースや青空文庫、Project Gutenberg のサイトで公開されている資料のように、特に言語研究を意識せずに電子化された広義のコーパスであれ、基本的に文字データのみで構成されている。したがって、コーパスは、特別なソフトでないと処理できないバイナリーファイルではなくテキストファイル(文字データだけで構成されたファイル)で構築するのが都合が良く、多くの場合、そのようになっている。テキストファイルであることによって、特別なコーパス分析用ソフトを用いる必要性から解放され、テキスト処理ツールなどで自在に処理することが可能になる。(現代英語の British National Corpus であれ、現代日本語の『現代日本語書き言葉均衡コーパス』(Balanced Corpus of Contemporary Written Japanese)であれ、テキストファイルであるから、利用環境や利用ソフトが限定されることはない。)

しかし、テキストファイルであるからといって、その処理を専らテキスト処理ツールなどで行っている限り、巨大なコーパスを対象にした複雑な処理を、言語研究の遂行に支障がない程度の速さで行うことが困難な場合も起こりうる。今後、コーパスの規模が巨大化することは明らかであり、巨大化すればするほどその困難さは増すことになる。そこで、本稿では、まず 2 節において基本的なテキスト処理ツールでできることとその限界について述べる。(その際、処理の明示化・透明化の必要性についても言及する。)3 節では、言語研究において有益ないわゆる picture について述べ、巨大なコーパスから picture の作成を高速に行うには、テキスト処理ツールでの処理では現実的ではないことを指摘する。それを踏まえ 4 節から 6 節においては、リレーショナルデータベースを用いてコーパスから有益な情報を高速に抽出するシステム(第 1 著者(西村)による開発)について、その設計方針を論じる。

2. grep による例文抽出と簡易な KWIC の作成およびその問題点

言語研究のためのコーパス利用で最も基本的な操作の一つは、ある特定の語(あるいは語句)を含む文の抽出である。そのためには、テキストを予め 1 行 1 文に整形しておき(あるいは予めその形式になっているテキストを対象に)、grep を用いて以下のような処理を行うだけで良い。ここでの検索対象は abundantly である。(以下、本稿では入力ファイルを in_file と表示する。)

(1) grep -Pi "\babundantly\b" in_file

grep は行単位での抽出を行うので、1 行が 1 文になっていれば、(1)のようなすこぶる単純な処理で文単位での例文抽出が可能となる。(ここで、\b は語の境界を示す正規表現である。また、オプション i を用いていることで、大文字・小文字の区別なく検索される。)

(2)は、abundantly の右隣に生起する語の頻度を調べるための処理である。

(2) grep -Pio "\babundantly [a-z-]+" in_file | tr '[A-Z]' '[a-z]' | sort |
uniq -c | sort -rn

ここでは、概略、abundantly という語の右隣にある語を abundantly と共に抽出し、大文字を小文字に統一して、各連鎖の頻度を降順で示す処理をしている。

(1)や(2)に比べると処理が複雑になるが、KWIC（Keyword in context。検索語を画面の中央に配置し、1行に入る限りで前後の文脈を示した形式）を作成するにも、プログラミング言語（ここで用いているのは Perl）によるいくつかの処理と並べ替えの機能をもつシェルコマンド sort による処理を組み合わせることで容易に行いうる。（もちろん、やや難度は増すものの、全ての処理を Perl だけで行うことも可能である。）KWIC は、言語研究上、語と語の共起関係やパターンを見るために多用される形式である。以下では、Project Gutenberg で公開されている Charles Dickens 著 *David Copperfield* のファイル（http://www.gutenberg.org/files/766/old/cprfd10.txt）を対象に bright という語の KWIC を作成している。

(3) a. 1 段落 1 行に整形
perl -pe 's/\r\n/ /; s/ /\n/;' cprfd10.txt |
b. bright を中心して、前後 30 文字（記号、空白文字も含む）の文脈を提示
perl -ne 'while(/\bbright\b/gip){printf "%30.30s|%s|%-30.30s\n",
substr(${^PREMATCH},-30), ${^MATCH}, ${^POSTMATCH};}' |
c. bright の右隣の語でソート
sort -t "|" -k 3,3

a, b, c をこの順番に実行することによって bright の KWIC が作成される。以下は、その一部である。

```
...
who had been used to wind her |bright| curls round and round her fin
hair, I watch her winding her |bright| curls round her fingers, and
 shepherd voices; but, as one |bright| evening cloud floated midway
e always is, a pattern, and a |bright| example.
dy still; and well I knew the |bright| eye with its lofty look, that
itself, at the moment, in her |bright| eye, that it came into my hea
ed that she had a very quick, |bright| eye.
  'Why, what a sight for such |bright| eyes at midnight!' I replied.
e ago, Doady,' said Dora, her |bright| eyes shining very brightly, a
er head, turned her delighted |bright| eyes up to mine, kissed me, b
t is; there, I meet a pair of |bright| eyes, and a blushing face; th
o me with tears of joy in her |bright| eyes, and said I was a dear o
at it gave me pain to see her |bright| face clouded - and for me! -
h enough of wilfulness in her |bright| face to justify what I had he
 looked up, and met her sharp |bright| glance respectfully.  'I have
d people all to nothing!' His |bright| glance went merrily round the
...
```

(3)では、文字単位で処理しているため、行頭および行末の語が完全に示されていない箇所があるが、bright が画面の中央に配置され、その右隣でソートされている様子が見て取れる。語の共起関係やパターンの把握によく使われるこの形式も、(3)のようなごく短い処理で行えるということである。

ここでは(3)の方法によりごく簡易な KWIC を作成した。KWIC の作成には、コーパス分析に特化されたソフトが利用されることが多いが、上記のような処理をすることには意味がある。それは、途中の処理過程がブラックボックスにならず、明示化・透明化されるという点である。これによって、処理の妥当性を他者が検証することが可能になる。この点は、研究目的でのコーパス利用には、格別重要なことである。また、このような方法を取れば、必要に応じて検索条件や出力形式などに部分的修正を加えることができるので、自由自在なテキスト処理が行い易くなるという利点もある。

しかしながら、(1)～(3)のような方法は、簡易ではあってもテキスト全体を 1 行ずつ検査するので、データ規模に比例して処理時間が長くなり、巨大なコーパスを対象にした場合、実用性に欠けるという問題がある。必要が生じた場合にはこの種の処理ができるようになっていることはコーパス利用上の基本的素養として必要ではあるが、膨大な時間がかかるようでは研究遂行上、実際的とはいえない。今後、コーパスが巨大化していくことは明らかなので、研究の遂行を妨げない程度に巨大なデータを高速に処理する手立てが求められるということになる。この点は、次節で見る picture を作成しようとする際に、より顕著に浮かび上がってくる問題である。

3. いわゆる picture の有用性

言語研究に用いられる形式に、いわゆる picture がある。picture とは、キーとなる語の前後に現れる語を頻度や MI-score（相互情報量）などの統計値の順に示したものであり、筆者らの知る限り、Collins COBUILD 英語辞典が依拠している Bank of English（現在の総語数は 6 億 5,000 万語のコーパス）のサイトで初めて実現したものである。（picture については、滝沢(1998)でも触れている。小学館コーパスネットワークにも picture 相当の機能がある。また Hunston (2002: 77-78)や齊藤・中村・赤野 (2005: 211-216)も参照。）

例えば、以下は、Bank of English（利用時の総語数は 5 億 2,000 万語）で instantly という語を対象に作成した頻度順の picture である。（一部、語の末尾が欠落しているのは、このシステムでは表示できる上限が1語あたり 10 文字までという制約があるからである。なお、NODE は、検索語である instantly が現れる位置である。）

the	the	was	NODE	recognisab	the	the
and	and	and	NODE	and	a	a
to	he	died	NODE	the	to	of
of	was	almost	NODE	to	by	was
that	to	is	NODE	<p>	and	to
a	it	killed	NODE	when	as	and
it	i	be	NODE	in	it	in
he	be	it	NODE	i	was	s
<p>	she	him	NODE	he	that	it
in	you	are	NODE	she	he	with
s	that	would	NODE	that	in	he
his	were	will	NODE	as	<p>	that
can	would	he	NODE	it	his	as
i	of	were	NODE	recognizab	with	his
you	they	knew	NODE	by	i	her
would	is	can	NODE	a	you	i
she	will	to	NODE	with	into	on
her	her	that	NODE	but	she	is
on	a	i	NODE	into	of	<p>
they	his	an	NODE	from	from	for
was	can	said	NODE	became	for	by
be	killing	you	NODE	forgettabl	up	had
which	s	not	NODE	recognised	their	you

この picture を見ると、instantly という語の共起関係についていくつかの特徴を読み取ることができる。（語末に欠落部分がある場合は、適宜、補って示すこととする。）

・NODE の右１語を見ると、recognisable（1 行目）と recognizable（14 行目）があることから、instantly recognisable/recognizable「すぐに分かる」というコロケーションが存在する。他にも instantly

recognised(23 行目)という連鎖も読み取れる。

・左 1 語に knew(15 行目)も(1)と同類である。

・左 1 語を見ると、died(3 行目)、killed(6 行目)があることから、died instantly や be killed instantly など「即死」に関係する語との連鎖が目立つ。また、左 2 語の killing(22 行目)も同類である。

・左 1 語の位置には almost(4 行目)という副詞が現れており、「ほとんどすぐに」を表している。

もう一つ別の例を見る。以下は「abundantly＋形容詞」という 2 語の連鎖の picture である。なお、この場合、NODE は abundantly が現れる位置である。

it	it	is	NODE	clear	that	the
<p>	made	it	NODE	plain	in	many
always	is	was	NODE	available	to	was
the	s	become	NODE	true	he	she
of	making	now	NODE	evident	the	this
in	make	be	NODE	fresh	<p>	a
s	this	made	NODE		and	is
was	way	all	NODE		is	first
but	are	becoming	NODE		on	won
that	that	t	NODE		now	more
things	had	still	NODE		with	which
should	should	himself	NODE		when	that
they	what	hellip	NODE		but	not
yet	may		NODE		you	comes
safety	after		NODE		letters	we
we	wasn		NODE		cbi	history
call	laws		NODE		hellip	ronald
speaking	debate		NODE			everyone
released	sophistica		NODE			magnificen
procedure			NODE			activists
topic			NODE			confronted
element			NODE			papua
theater			NODE			romance

この picture からは以下のことを読み取ることができる。

・右 1 語を見ると、clear が 1 行目に来ており、abundantly と共起する形容詞としては clear が最も高頻度である。

・右 2 語には that が 1 行目にあり、左側には it, is, become, becoming, made などが目立っているので、ここから it is abundantly clear that ...や it was made abundantly clear that ...、it is becoming abundantly clear that ...などのパターンが浮かび上がってくる。さらに、左 1 語に now(5 行目)があることから、it is now abundantly clear that ...というパターンも存在する。

このように前後に現れる語の分布の鳥瞰図を得ることを可能にしているのが picture であり、言語研究とりわけコロケーションやフレイジオロジー(八木・井上 (2013)など)の研究において有益であ

る。

以上、Bank of English によって例示したように、既存のコーパス検索システムには picture を作る機能が整備されているものもあるが、完全にブラックボックスになっており、どのような方法でこの機能を高速に実現しているのかに関して利用者が知ることができるようにはなっていない。また、この形式を作成することは、前節で見たような簡易な方法で行うことはできず、その処理はかなり複雑なものとなることが予測される。仮に複雑な処理によって picture を作成したとしても、1 行ずつ検査している限り、処理時間の問題から逃れることはできない。このような picture を大規模なテキストからストレスを感じない程度の時間でどのように作成するかは、コーパス研究上の重要な課題である。

そこで、筆者らは、リレーショナルデータベースによってこの機能を実現することを企画し、言語研究者の必要を十分に満たすシステム開発を行った。以下では、開発したシステムの基本的設計について論じることとする。

4. リレーショナルデータベースの利用

4.1 データベース型の種類

データベースの種類について論じた國友（1994: 216-225）を要約すると、次のようになる。データベースの型は階層型データベースと関係データベースの 2 つに大別できる。前者はデータを階層型、すなわち親子関係の構造でとらえたデータベースであり、子データへのポインターを親データに記録させ、親データと子データをまとめて一つの意味を成すデータとして扱う。これは、データ項目重複の最小化や処理効率が良いという利点があり、企業における基幹システムのように、情報への要求が固定しているようなシステムで従来から多く用いられてきた。しかしながら、構造が定型化することにより柔軟性に欠けるきらいがあるという短所も併せ持っている[2]。一方、関係データベースはまとまりのあるデータを表（以下テーブルと呼ぶ）にまとめ、データをその中に列として記録することで柔軟性に富むことから、情報に対する要求の多様化に伴い最近では関係データベースが主役になりつつある。以上が要約である。

関係データベース（以下リレーショナルデータベースと呼ぶ）の比較的身近な例を挙げると Microsoft 社の表計算ソフト Excel やデータベースソフト Access がある。非常に柔軟性のあるデータ構造である。また、一般に操作言語（DML: Database Manipulation Language）には SQL（Structured Query Language）と呼ばれる比較的簡単なプログラミング言語が使用されることや、リレーショナルデータベースマネジメントシステム(RDBMS: Relational Data Base Management System)には商用の Oracle Database や IBM DB2、オープンソースの MySQL や PostgreSQL などが容易に利用できることとも相まって、現在ではリレーショナルデータベースが主流になっている[3]。

このような背景から、コーパスのデータベース化にリレーショナルデータベース技術を採用し、RDBMS には MySQL を利用することにした[4] 。システム全体のプログラミング言語には Perl を用い、MySQL とのインターフェース設計には Descartes and Bunce (2000) を、MySQL を効果的に

活用するためには Schwartz *et al.* (2008) をデータベース設計の参考にした。

4.2 リレーショナルデータベースのイメージ

リレーショナルデータベースは、複数のテーブルを関連付けて操作するデータベースである。テーブルは行（row）と列（column）で構成され、テーブルを同じ属性を持つデータの集まり、すなわち集合とみなして、テーブルまたはテーブル間の関連付けを集合論的に操作する。ここで、架空の「学生テーブル」（学籍番号,氏名,出身高校名,指導教員名）をもとに集合論的操作である射影（projection）、選択（selection）、結合（join）および商（division）について述べる。なお、カッコ内はテーブル内の列名を指している。このテーブルから、氏名と指導教員名の列を取り出す操作が射影、指導教員名を指定して、この教員に指導を受ける学生の行を取り出す操作が選択に相当する。次に、「出身高校テーブル」（出身高校名,都道府県）を仮定し、このテーブルを使って「学生テーブル」に（出身高校の）都道府県を追加したテーブルを作成する操作が結合である。つづいて、「学生テーブル」から複数の出身高校を指定して出身学生を抽出することを想定する。この場合、出身高校名を行とするテーブルを準備して、このテーブルで「学生テーブル」から該当する学生の行を抽出する操作が商である。

5. データベースの設計と構築プログラム

5.1 設計から見た言語テキストの要点

そこで、リレーショナルデータベース技術を用いてコーパスデータベースを実現するために、言語テキストのどのような特徴に着目すべきなのかを検討する。

a. ファイル -> パラグラフ -> センテンス -> 語という分節性に着眼する。しかしながら、ファイルそのものはデータベース内に存在する必要はない。

b. 「ファイル名」は抽出データの出典を示すための必須情報である。ただし、これのみでは元のテキスト位置の特定に不便なので、「ファイル名＋パラグラフ識別コード」の形式で出典索引情報としてデータベースの中に記録する。

c.言語研究のためには、語にはそのレマと品詞情報がある方が有益である。一般的には、データベースの元のテキストにはレマと品詞情報が含まれているとは限らないので、必要に応じてコーパスデータベースの作成過程で付与しなければならない。

d. 記録した全テキストを対象に、語（およびレマ、品詞情報）を唯一の検索キーとして、この検索キーを含むセンテンス（またはパラグラフ）を抽出できることが必要条件である。

e. したがって、検索キーをもとに最も効率の良い検索アルゴリズムと、これに対応するデータベース構造を設計することが求められる。

以下にデータベース構造案（試作モデルも含めて）とその評価を述べる。なお、出典テキストの所在を示す「ファイル名＋パラグラフ識別コード」は以下 fnosno と呼ぶ[5]。

5.2 データベース構造設計の要点とコーパスデータベース

ここで、データベースの論理構造設計の要点を整理したうえで、コーパスデータベースの特徴を述べる。

(1) データベース構造設計の要点

一般的に、データベース設計においては、事前に次の事項を検討することが重要である。

a. 正規化

リレーショナルデータベースはテーブルがデータの集合体として 1 つの単位となり、複数のテーブルを関連づけることによって、効率よくデータを管理する。テーブル設計においては「正規化」という作業によりデータ重複を最小にし、テーブル内のデータ修正時の誤り発生を極力避ける。たとえば、先に仮定した「学生テーブル」に学生の所属学科名を追加し、(学籍番号,氏名,出身高校名,指導教員名,所属学科名)とすることを想定して説明する。この場合、「学生テーブル」内の一人一人の学生に対して所属する学科名を入力していくと、入力ミスが危惧される。さらに、学科再編により学科名が変更されると同じ作業のやり直しが必要になり、ここにもミス誘発の原因が内在することになる。これを回避する一方法として、「学科テーブル」(学科コード,学科名)を作り、「学生テーブル」には学科名でなく、学科コードを入力しておく。将来、学科名変更があれば、「学科テーブル」のみ改訂すればよい。正規化とはこのようなテーブル設計作業を言う[6]。

b. トランザクション処理と排他制御

トランザクション処理とは、関連するテーブル(群)の更新処理を伴うデータが発生したときに、データの受付から全処理を完結するまでのひとまとまりの処理である。この時、複数端末から入力されたデータが偶然、同じテーブルの同一行を照会(すなわち、処理競合が発生)すると、結果的に矛盾が発生、もしくは処理続行不能になる事態が発生する[7]。これを防ぐために適切な排他処理が必要になる。

c. リカバリー対策(ロールバック)

何らかの障害発生でシステムが停止し、その後再開するときに、停止直前の処理は中途半端な状態にあるので、これを初期状態に戻して(ロールバック)、処理を再開させる対策が必要になる。

(2) コーパスデータベース設計の特殊性

コーパスデータベースの特徴は、まず対象データは(品詞,レマ,語)の集まりであり、これらは正規化の対象とはなりえないので、正規化について考慮する必要はない。なお、品詞は POS タグとしてコード化されているので(POS タグ,品詞名)テーブルが必要な場合があるが、レマと語はデータそのものである。

次に、いったん構築されたデータベースへの更新・追加・削除はなく、検索のみに利用される。すなわち、トランザクション処理が行われることはない。したがって、一般のデータベースのように排他制御やリカバリー対策を考慮する必要性は低い[8]。さらに、大量のデータを検索するコーパスデータベースでは、検索速度の速いストレージエンジンが望ましい。これらの特徴を考慮して設計を進めることになる。一般のリレーショナルデータベースと比較すると、コーパスリレーショナルデータベースは特異なデータベースといえる。

検索処理に関しては、MySQL には合計 8 種のストレージエンジンが提供されている。代表的なエンジンは MyISAM と InnoDB の 2 種類である。前者はトランザクション処理のサポートはないが高速に動作、後者はトランザクション処理とリカバリーに対応している。このことから、ストレージエンジンには MyISAM が適切と判断し、これを採用する。

5.3 British National Corpus (XML Edition) に収録された語の統計

以上を踏まえて、5.4 節～5.7 節は British National Corpus (XML Edition)（以下 BNC と略記）をソースデータとして、コーパスデータベース構築モデル開発の考え方と実情を述べる。これに先立って、本コーパスに含まれる lemma および word の統計情報について説明する。なお、これ以降はデータベース内および検索式などの具体的指示では、lemma および word で表記し、一般論としてはレマおよび語と表記して区別する。

表 1 BNC に含まれるレマの分布[9]

レマ当り度数	レマ数	度数 (x1,000)	割合 (%)	累積割合 (%)
≧100,000	92	49,967	50.8	50.8
99,999 - 50,000	92	6,116	6.2	57.0
49,999 - 10,000	902	17,284	17.6	74.6
9,999 - 5,000	875	7,123	7.2	81.8
4,999 - 1,000	4,032	8,817	9.0	90.8
999 - 500	3,347	2,363	2.4	93.2
499 - 100	15,656	3,479	3.5	96.7
<100	580,215	3,169	3.2	99.9
計	605,211	98,318		

表 2 BNC における度数上位 10 のレマ[10]

レマ	度数 (x1,000)
the	6,042
be	4,120
of	3,041
and	2,617
to	2,593
a	2,164
in	1,938
have	1,317
it	1,215
he	1,198

5.4 最も単純な構造試案（第 1 次システム）

本構造は、語が含まれるパラグラフを検索するのみの最も単純な構造である。

(1) テーブル構成

図 1 を参照。

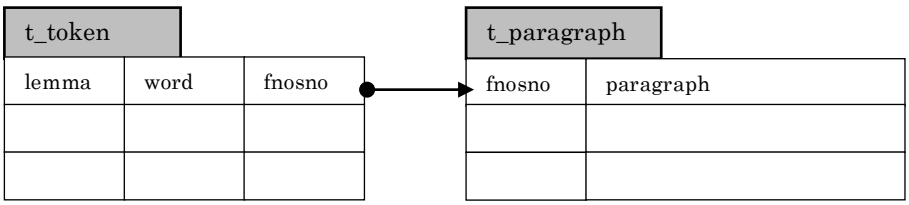

図 1　最も単純なデータベース構造案（第 1 次システム）

1) パラグラフのすべての語でテーブル t_token を作成する。

2) t_token 内のフィールド fnosno が t_paragraph の fnosno に関連付けられている。

(2) 処理手順

1) 検索 word（または lemma）に一致する行を t_token から select（抽出）する。

2) 同一検索 word で fnosno の重複があるのでこれを一つにまとめてテーブルを作成する。

3) t_paragraph から fnosno の一致するパラグラフを抜き出す。

4) （後処理で）コロケーション、KWIC、picture などに整形する。

(3) 評価[11]

×	a. 2 節の(1)(2)(3)式の処理をデータベースに置き換えただけで、処理時間も長くなる。
×	b. 複数語検索や品詞タグ指定を伴うと後処理に回される。
×	c. 統計資料の基になる lemma と word の度数がデータベース内にない。
○	d. 構造が単純なのでデータベースに対する操作はわかりやすい。

5.5 lemma と word の頻度テーブルの作成と fnosno との対応付け（第 2 次システム）

(1) 改良の着眼点

1) lemma と word の度数を記録するテーブルを設ける（k_lemma、k_word）。

2) t_token を tkn_[k/u]$_i$ というテーブルにグループ分けして検索対象範囲を狭める。これにより処理時間の短縮を図る[12]。

3) グループは lemma 単位（もしくは lemma 群）にし、k_lemma テーブルと関係づける。

4) プレーンテキスト t_paragraph を品詞タグと lemma 付き（'>POStag>lemma>word>'フォーム）の t_tag に変更し、応用範囲を広げる。

(2) テーブル構成

図 2 に改良案を示す。

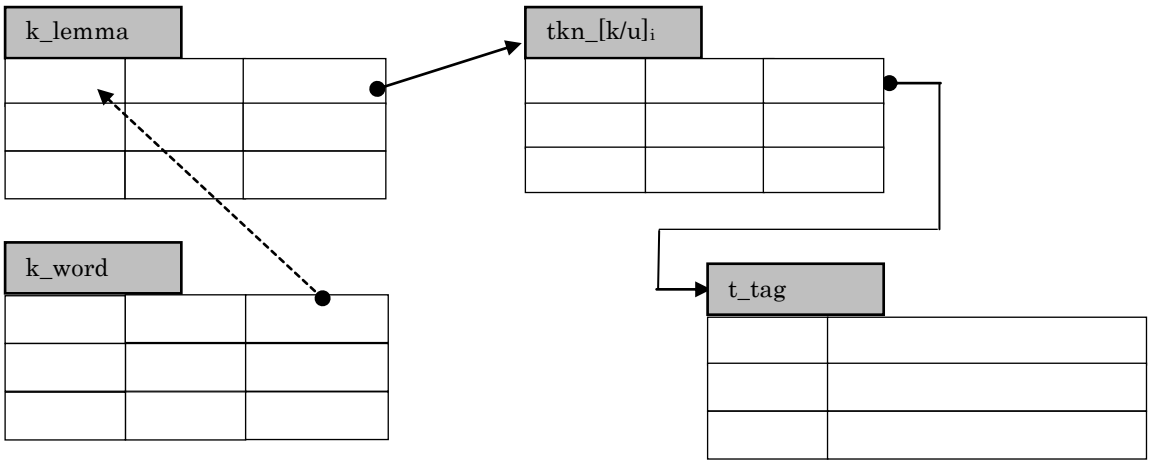

図 2 度数表とトークンテーブルを追加(第 2 次システム)

(3) 処理手順

1) 検索語が word の場合は k_word -> k_lemma とたどり、検索語が lemma の場合は直接 k_lemma から対象となる $tkn\_[k/u]_i$ を得る。

2) 検索語が複数ある場合は、各 lemma の度数(freq)を比較し、最小の lemma を選定してその $tkn\_[k/u]_i$ を得てそのテーブル内から fnosno を抽出して集合を作成する。

3) 作成した集合を、fnosno の重複を一つにまとめたテーブルに編成する。

4) t_tag から fnosno の一致するパラグラフを抜き出す。

5) (後処理で)コロケーション、KWIC、picture などに整形する。

(4) 評価

×	a. 対象となる fnosno の抽出は 3.4 節の処理よりより格段に速くなるが、ヒット件数が多い場合は MySQL のオーバーヘッド分、2 節(1)(2)(3)式と比較して処理時間は長くなる。
△	b. 検索対象が $tkn\_[k/u]_i$ に絞りこまれている分、検索時間は短縮される。
○	c. 統計資料の基になる lemma と word の度数が記録されている。

5.6 Ngram テーブル(第 3 次システム)

(1) 改良の着眼点

第 2 次改良システムは、検索語から対象テキストを絞り込む処理は高速化できるが、さらに続けてテキストテーブル t_tag を検索しなければならない。この 2 段階検索を 1 段階検索で済ませることができれば処理速度を短縮できる。

一方、コロケーション、KWIC、picture は検索語を中心(NODE)にある語数を加えたテキスト、すなわち ngram のテキストがあれば得られる。このことに着目し、ngram の n 数を考察すると、コロケーションではほぼ NODE の前後 2 要素(n=5)、KWIC では NODE の前後 10 要素(n=21)程度、picture で NODE の前後 5 要素(n=11)あれば研究に資すると仮定した。このことから、$tkn\_[k/u]_i$

に 21gram のテキストを追加することで、2 段階検索を回避することにする。

(2) テーブル構成

図 3 を参照。

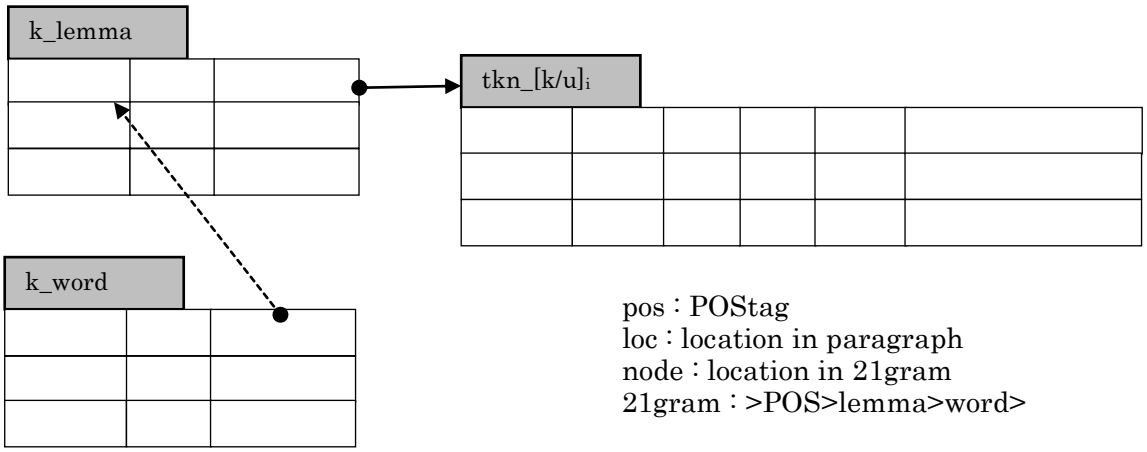

図 3 最終構成図（21gram の作成）

(3) 処理手順

1) 検索語が word のときは、k_word から lemma を得る。

2) k_lemma を lemma で検索し tkn_[k/u]$_i$ を得る。

3) テーブル tkn_[k/u]$_i$ から 21gram を得る。この時、

a. tkn_k$_i$ であればテーブル内全行が対象となる。ただし、検索語が word であれば word で対象を絞り込む。

b. tkn_u$_i$ であればテーブル内から lemma（検索語が word であれば lemma と word）で絞り込む。

4) 後処理で、コロケーション、KWIC、picture に整形する。

(4) 評価

×	a. 21gram により記録データが重複する[13]。これはデータベースの基本的な考え（正規化によってデータ重複を極力少なくする）に反する。
○	b. しかし処理速度は満足できる水準になる（6 節参照）。

5.7 テーブルのクラスタリング（第 4 次システム）

本テーマは技術的な問題なので簡単に述べる。

21gram に着目した第 3 次システムのデータ構造では、検索対象データが論理的に一つのテーブルに集約されている。しかし、コーパスデータベースのように大容量のデータベースでは、格納テーブルの物理的クラスタリングが処理効率を決定づけると言える。

(1) クラスタリング

図 3 が論理的なデータ構造であるのに対して、データを HD に記録する場合（ハード的）にテー

ブルのデータを近傍に配置することをクラスタリングという。HDD は回転盤上の同心円（トラック）にデータを記録するので、データ転送には HDD から内部メモリーへの転送時間に加えて、トラック移動時間（データ読み書き用ヘッドの移動時間：シークタイム）と回転待ち時間が加算される。このため、シークタイムと回転待ち時間の合計時間を最小にするようにデータを近傍に集中格納して（クラスタリング）、レスポンス時間の短縮を図る。

(2) クラスタリングの失敗と改良

データベースのビルド処理において、ソースデータから図 3 のように順次データを格納すると、各テーブルが広範囲のトラックに分散して記録されクラスタリングに失敗する。第 3 次システムによるレスポンスタイム実測値が非常に長いことが判明したので、クラスタリングの失敗を想定して対策を検討した。その結果、ビルドプログラムではテーブルに格納するデータを一旦ファイルに蓄積して、最後にまとめて各ファイルを順次データベースのテーブルにインポートするように変更したところ、処理速度が格段に短縮された。

(3) 改良後の処理時間

3節で、pictureを大規模なテキストからストレスを感じない程度の時間で作成することには、研究上、有益である、と述べたが、第 4 次システムにおいて、1億語規模のコーパスから度数 10,000 の単語の picture を 1 秒以内で作成することが確認できた。アプリケーションプログラムの処理時間は 6.8 節にまとめて報告する。

5.8 BNC 以外のソースデータへの適用

前節までは BNC からのデータベース構築であるが、普通のテキストファイルからのコーパスデータベース構築が目的なので、コーパスデータベース構築システム（ビルダー）を普通のテキストファイルに適用できるように改良する。これには次の 3 点を考慮する必要がある。

a. tagging と lemmatizing

BNC は予め POS タグと lemma が付加されたテキストである[14]。しかし普通のテキストをソースとする場合は、何らかの手段を講じて tagging と lemmatizing が必要となる。本稿執筆段階ではシュトゥットガルト大学が公開している TreeTagger（入力テキストに対して、品詞タグを付加するとともにレマも付加する）をビルダーに組み込んである[15]。

b. テキストファイルの文字コードと多言語対応

BNC の文字コードは UTF-8 である。TreeTagger はソースに Latin1 を指定しているので、開発したビルダーを使用して英語のコーパスデータベースを作成するためには、文字コードは Latin1 という制約を受ける。UTF-8 などの Latin1 以外の文字コードのテキストの場合には、文字コード変換が必要である。

c. ソースファイルに求められる条件（英語テキスト）

段落を示す<p>タグを該当箇所に書き込んでおけば、ビルダーは<p>([^<]+)<で示すカッコ内をパラグラフテキストとしてデータベースに取り込む。構築に際しては事前にこの加工処理を実施しておく必要がある。

5.9 コーパスデータベース構築システム(ビルダー)の最終版

以上をまとめて、ビルダーのプロセスイメージを図 4 に示す[16]。

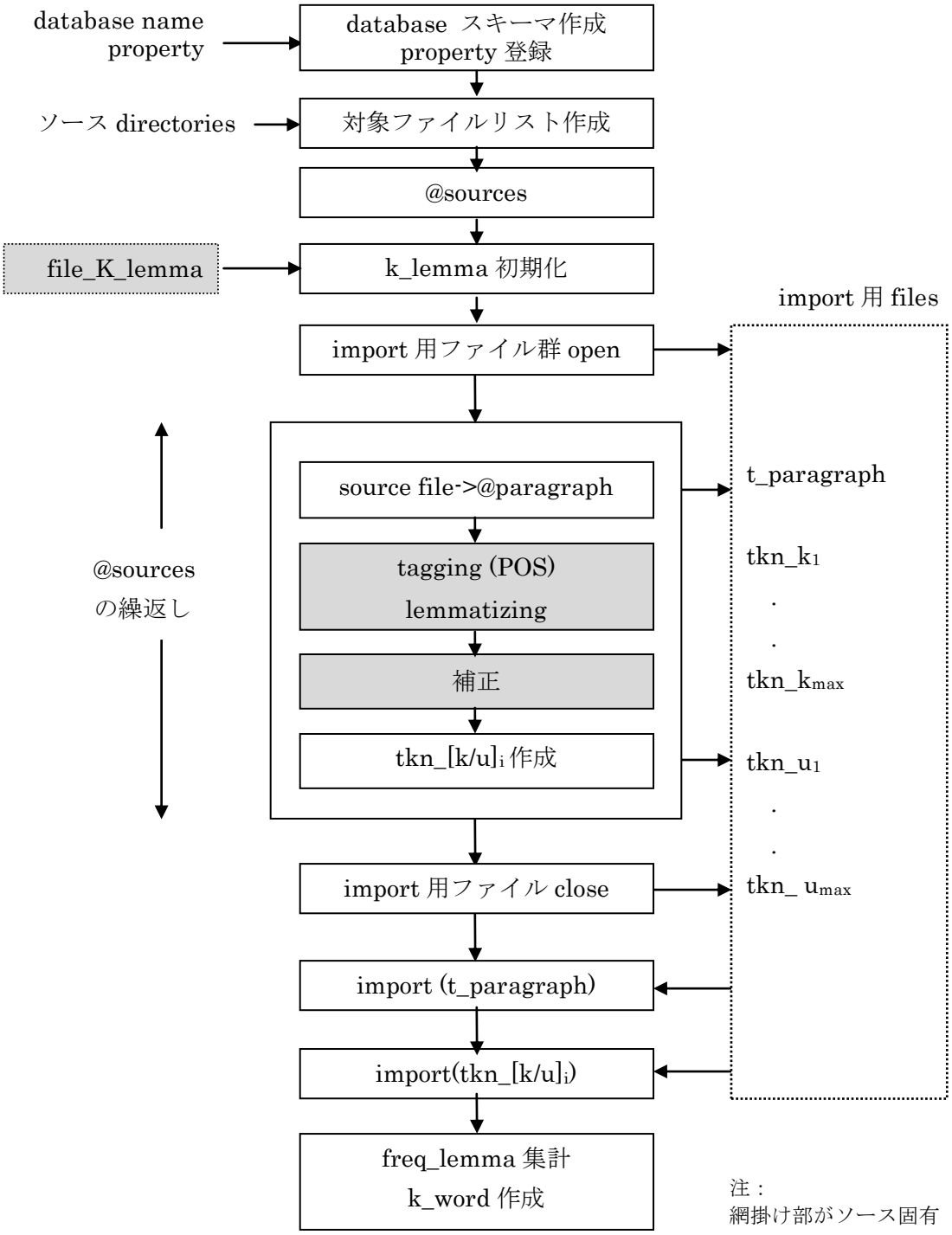

図 4 コーパスデータベースビルダーのプロセス

処理開始直後の「property 登録」は、新規に作成するデータベースのソース名、年代、ジャンル、

地域などのデータを管理テーブルに記録する、いわばデータベースのデータベースを作成する処理である。アプリケーションプログラムはこの管理テーブルを利用して対象データベースの選択、あるいは組み合わせを指定できる。

なお、ビルダーは次の 3 種類で、この内 build_db_jpn は今後、開発する予定である。

表 3 コーパスデータベースのビルダー

名称	ソーステキスト
build_db_bncx	BNC
build_db_eng	ASCII または Latin1 の英語テキスト。TreeTagger を利用する。
build_db_jpn	日本語テキスト。（開発予定）

6. アプリケーションプログラムと実行例

本稿執筆時に完成しているアプリケーションプログラムと、BNC をもとに作成した db_bncx を対象に処理したサンプル画面を示す。使用した PC サーバーは Dell PowerEdge T110（インテル Xeon プロセッサーX3434（2.40GHz, 8MB キャッシュ））で、オペレーティングシステムは Fedora14 である。なお RDBMS は MySQL5.1.60、アプリケーションプログラムの言語は Perl v5.12.4 である。

6.1 アプリケーションプログラムの構成

アプリケーションプログラムは Linux の端末画面で操作する。起動プログラムで処理選択が指示され、コードを入力すると当該アプリケーションプログラムが起動する。アプリケーションの種類を表 4 に示す。

表 4 アプリケーション一覧

コード	処理	説明
r	lemma、word の度数	lemma と word（屈折形）の関係、度数を一覧する
c	コロケーション	検索語の左右、品詞タグを指定
ks	KWIC 出力（1 語指定）	出力のソート条件を指定
km	KWIC 出力（2 または 3 語指定）	検索語間の距離も指定できる
p	picture	範囲は L5～R5、度数順に 20 行[17]

各アプリケーションプログラムは操作者に検索条件入力を促し、処理結果をファイルに出力する。ただし、picture のみほぼ端末画面に収まるので端末に表示する。表示最下段で次に使用するアプリケーションプログラムを指定し、連続作業を可能にする。6.2 節～6.7 節で検索条件と出力サンプル、および処理時間実測値を示す。図 5 はアプリケーションプログラム起動直後の画面である。

```
***********   Welcome nishi san to applications for corpus databases   ***********

Applications:
                        d   -> choose and set database name
                        r   -> refer lemma/word information (lemma/word, frequency, and t_tkn_[name])
                        p   -> picture screen
                        c   -> collocations
                        ks -> kwic with single query
                        km -> kwic with multi query
                        q   -> quit
using database ==> ## db_bncx db_nants db_apws ##

********** ==> Type in application code(d, r, p, c, ks or km) or q(uit)
```

図 5　アプリケーション初期画面

6.2　検索式のシンタックス

アプリケーションプログラムでは検索条件を次の書式で指定する。

(1) lemma と word の区別

lemma:*string*　または *string*>

word　:>*string*

(2) 任意の文字列は'*'で代用

正規表現の'[^>]*>'に相当する。たとえば'*ly'は語尾に'ly'を伴う任意の語を意味する。

(3) 区切り記号'#'

複数検索語のときに、検索語間の距離指定に使用する。

(4) 位置情報'L/R(n)'

検索語に対して左側/右側の要素を指定する。n は検索語からの位置。

(5) 検索語の構成は POS>lemma>word

POS タグのみ指定するときは、たとえば AJ0>*>*と指定する。

6.3　語彙情報参照（コード='r'）

データベース内の lemma もしくは word の統計情報を検索する。図 6 は lemma='enable' の情報を参照した例である。lemma の度数は 10,003、さらに 4 種の word 別度数がわかる。

```
----------> Type in key word (format: lemma>, or lemma, or >word)
enable
------
database name = << db_bncx >>
        lemma          word              freq        tkn_name
        enable                          10003       tkn_u186
                       enable            4737
                       enabled           1803
                       enables           2098
                       enabling          1365
```

図 6　lemma=’enable’の情報を参照

次に word=’building’で参照した例を図 7 に示す。lemma ‘build’ 内の屈折形’building’も参照していることがわかる。

```
----------> Type in key word (format: lemma>, or lemma, or >word)
>building
---------
database name = << db_bncx >>
            lemma              word                  freq       tkn_name
            build                                   23072       tkn_u54
                               build                 7125
                               builded                  4
                               building              2725
                               builds                 573
                               built                12645
            building                                22524       tkn_u77
                               building             15956
                               buildings             6568
            ------------------------------------------------------------
```

図 7　word=’building’の情報を参照

lemmatizing は完全に正しいわけではないので、検索作業を始める前に検索語の情報を事前に把握し、検索語を必要に応じて lemma と word で使い分ける必要がある。

6.4 コロケーション（コード=’c’）

コロケーションを求める条件を、検索語に対する共起位置、および共起語の属性（主として品詞）を考慮して指定する。検索式は図 8 中の Sample1 および Sample2 のように例示してある。図 8 で実行した検索式 “because#L1#*AV*>*>*”は、接続詞 because の直前に生起する副詞の度数をカウントすることを指示している。

```
==========> Type in query (format: lemma(lemma>,>word)#[LR]n(#pos>lemma>word}
        Sample 1：absolutely#R1
        Sample 1：absolutely#R1#*JJ*>*>*
because#L1#*AV*>*>*
        ------------------------------------------------------------
        **Result**  database = <db_bncx>     568keywords  14490matched  elapsed time
1.98sec
        output file --> /home/nishi/rslt_db/col/because_L1_AV.txt
        ------------------------------------------------------------
```

図 8 コロケーション

結果出力ファイル because_L1_AV.txt から高度数 10 件を図 9 に示す。

```
partly     1271
just       1195
only       1023
simply      852
perhaps     586
also        451
mainly      378
largely     353
least       346
precisely   331
probably    320
```

図 9 接続詞 because の直前に共起する副詞

6.5 KWIC(ks)

本アプリケーションプログラムは 1 語のみ指定して抽出し、結果を KWIC 形式でファイルに出力する。図 10 は検索語として lemma="albeit"を指定した実行結果である。抽出件数は 1,382 件、処理時間は 0.52 秒である。（出力ファイルの内容は省略する。）

```
********** ==> Type in application code(d, r, p, c, ks or km) or q)
ks
==========> Type in retrieve key (format = lemma, lemma>, or >word)
albeit
==========> Type in sort position (L=left side of key, K=key, R=right side of key)
L
        ------------------------------------------------------------
        **Result**  database = <db_bncx>     1382rows matched  elapsed time  0.52sec
        output file --> /home/nishi/rslt_db/kwic/albeit.txt
        ------------------------------------------------------------
```

図 10 KWIC(ks) 処理画面（検索語は 1 語）

6.6 KWIC(mk)

本アプリケーションプログラムは検索式に最大3語まで与え、結果をKWIC形式でファイルに出力する。検索式は図 11 の冒頭で説明されているように、

pos1>lemma1>word1#d1#pos2>lemma2>word2[#d2#pos3>lemma3>word3]

で与える。ここで、「d1」は第 1 検索語と第 2 検索語の間に入る要素数、「d2」は第 2 検索語と第 3 検索語間に入る要素数である[18]。検索語が 2 件の場合は[…]を省略する。図 11 はイディオム'kick the bucket'検索の実行結果で、マッチしたのは 13 件、処理時間は 0.75 秒である。

```
********** ==> Type in application code(d, r, p, c, ks or km) or q)
km
==========>Type                                                    in                    querys
(pos1>lemma1>word1#d1#pos2>lemma2>word2[#d2#pos3>lemma3>word3])
                              '*' is replaced into '[^>]*'      d <= 5
*>kick>*#0#*>the>*#0#*>bucket>*
==========> Type in sort position (L=left side of key, K=key, R=right side of key)
L

        database=<db_bncx>
        lemma   kick      5161      tkn_u15
        lemma   the       6041815   tkn_k1
        lemma   bucket    1359      tkn_u484
        C3      <NODE> lemma    bucket

        ---------------------------------------------------------------------------
        13rows matched   elapsed time   0.75sec
        output file --> /home/nishi/rslt_db/kwic/kick_0_the_0_bucket.txt
        ===========================================================
```

図 11 KWIC(km) 処理画面（検索語は 2 または 3 語）

図中の「L」は検索式にマッチした語群の左側でソートして KWIC を出力することを指示する（右側であれば「R」、検索式にマッチした要素であれば「K」）。「C3 <NODE> lemma bucket」は、検索語が 3 件あり、この中の 3 番目の lemma すなわち bucket の 21gram が格納されているテーブル（=tkn_u484）から検索したことを示している。すなわち、3 検索語の中で最も度数の低い語に着目して、処理速度の短縮を図っている。

6.7 picture

本稿執筆時点で完成しているプログラムは、検索語（NODE）の左側 5 要素（L5..L1 まで）、右側 5 要素（R1..R5）までを対象に、各位置に生起する lemma の度数を集計し、度数の高い順に 20 番目まで端末画面に表示する。ここで、L5…L1 は NODE の左側の要素、R1…R5 は NODE の右側

の要素を表し、いずれも小さい数値が NODE により近いことを表す（L1 は 3 節で述べた左1語に、R1 は右1語に対応する）。検索語は lemma または word である。したがって、画面構成は 1 行が“L5...L1 NODE R1..R5”で構成され、これを 20 行表示する。

図 12 は lemma=”enable”の picture で、紙面の都合上“L3..L1 NODE R1..R3”および 5 行を切り出してある。図の上段が lemma の分布、下段がこれに対応する度数を示している。enable の度数は 10,002 であるが、この picture 作成時間は 0.70 秒である。

the	<PUN>	to	NODE	the	to	to
<PUN>	this	<PUN>	NODE	they	<PUN>	of
be	which	which	NODE	we	of	be
of	that	will	NODE	you	and	<PUN>
and	and	would	NODE	he	user	and
640	924	2275	NODE	2025	4754	2217
540	368	862	NODE	944	165	384
468	331	754	NODE	728	152	338
237	252	739	NODE	572	124	182
289	236	414	NODE	524	79	140

図 12　enable（度数 10,002）の picture

図 13 は 6.4 節で例示した接続詞 because の picture（L3..L1 NODE R1..R3/10rows）である。副詞 partly、just、only の位置がよくわかる。なお、処理時間は 5.43 秒である。

be	be	<PUN>	NODE	of	be	be
<PUN>	<PUN>	be	NODE	it	the	not
the	the	and	NODE	the	have	the
to	it	but	NODE	they	do	<PUN>
a	not	partly	NODE	i	it	a
of	to	just	NODE	he	<PUN>	have
it	this	it	NODE	you	they	to
in	and	not	NODE	we	can	of
not	that	that	NODE	she	a	that
have	of	only	NODE	<PUN>	would	it
5695	7205	22280	NODE	17644	19288	6960
4796	6992	3966	NODE	10743	8413	6122
3817	3479	1671	NODE	10113	6972	3677
3072	2144	1573	NODE	9412	2606	3138
2142	1957	1271	NODE	7190	1569	2794
2098	1931	1197	NODE	6526	1486	2730
1792	1881	1191	NODE	3539	1433	2385
1765	1460	1170	NODE	3357	1387	2133
1641	1366	1087	NODE	3194	1129	1346
1420	1357	1023	NODE	2981	1118	1207

図 13　because（度数 100,486）の picture

6.8 処理時間のまとめ

6.4 節～6.7 節で実行サンプルを説明したアプリケーションプログラムの処理時間実績を表 5 にまとめる。なお、処理時間とはプログラムがキーボードから検索式を受け取り、検索結果を端末画面に表示し終わるまでの時間であり、ほぼ応答時間に近似する。

表 5 アプリケーションの処理時間

アプリケーション	検索指定 lemma	Lemma の度数	処理時間（秒）
コロケーション	because	100,486	1.98
KWIC(1 語)	albeit	1,382	0.52
KWIC(3 語)	bucket	1,359	0.75
picture	enable	10,002	0.70
picture	because	100,486	5.43

情報システムの政府調達に係る SLA 導入研究会 (2004) はソフトウェア開発・調達におけるサービスレベル契約（Service Level Agreement : SLA）のガイドラインを示すものであるが、この中で基準応答時間達成率をサービスレベルの一つに指定し、次のように述べている。

> 利用者からの操作に対するシステムの応答時間を計測し、そのうち、基準応答時間である3秒以内にどの程度応答できたかを、サービスレベルとして設定する。1000回の操作のうち、2 回だけ 3 秒以上かかったとすると、基準応答時間達成率は、99.8% ({1-(2/1000)}x100) である。(p.7)

ここに示されている基準応答時間 3.0 秒以内を一つの基準値と仮定すれば、本アプリケーションプログラムはほぼ満足できる処理時間に収まっていると言える。検索語の度数に比例して処理時間が伸びるのは致し方ないが、度数 50,000 程度までの語であればほぼ 3 秒以内に picture を出力できる水準であり、言語研究を遂行する上でも十分な速度を達成している。

6.9 課題

これまでに現在完成しているアプリケーションプログラムの概要を述べ、実行結果画面のサンプルを示してきたが、次のような課題を残している。

a. tagger、lemmatizer

POStag 付けや lemmatizing されていない一般の英文テキストの場合、現時点では TreeTagger を利用しているが、さらに高精度な形態素解析ソフトを入手できると、コーパス解析の品質が向上する。

b. アプリケーションプログラム

必要と考えられる一通りの機能を持つアプリケーションプログラムを開発し実用段階にはあるが、特に picture 作成プログラムは改良すべき点がいくつかある。たとえば、集計は lemma に限定されているが、ある語に分詞が多く共起する場合は word を集計しなければならない。あるいは、picture 上の特定の共起語にカーソルを当てると、KWIC 形式で一覧表示することができれば、わざわざ KWIC(km)プログラムを起動させる必要がない。また、利用者の端末画面に適した picture 表示に変更すること（サイズ変更）なども望まれることになるだろう。さらに、k_lemma テーブルにはレマの度数、k_wordテーブルには語の度数が集計されていること、また総語数も集計できることから、t-score や MI-score を求めることも可能となり、これらの統計値を使用した picture 作成は今後の課題である。

c. 日本語への対応

日本語対応ビルダーを開発することで、コーパス利用分野が拡張される。現段階のビルダーの小規模な変更（具体的には文字コードの変更と、形態素解析器である 茶筌もしくは MeCab の組み込み）で実現可能である。

d. データ記憶装置

近年注目されている大容量のデータ記憶装置に、ソリッドステートドライブ（solid state drive : SSD）がある。これは記憶装置として半導体素子メモリーを用いたストレージとして扱うことのできるもので、HDD のようなシークタイムを必要とせず、ランダムアクセス性に優れている。将来的には HDD に代わるものとして期待される。その場合、5.7 節で述べたクラスタリングの制約が大きく緩和され、データベースの物理的格納方式が変わる可能性が高い。

e. NoSQL

4.1 節で 2 つのデータベース型について述べたが、近年、NoSQL（Not only SQL）というデータベース技術が注目を集めている。これは、リレーショナルデータベースがプログラミング言語 SQL によって操作されることに対して命名されているもので、構造化に向かないような大規模データ（たとえば、ブログやツイッター）から情報を取り出そうという目的で開発されている。数種のモデルが提案もしくは販売されており、中にはテキストに特化した製品もある。それらの中にコーパスに応用可能なシステムがすでに開発されているか、もしくは開発が計画されているかもしれず、今回開発したシステムの短所であるデータ冗長度が高いために、大容量記憶装置を必要とする点を改善できる可能性がある。現段階では未調査であり、今後、技術動向を注目していきたい。

7. 終わりに

本稿で述べたシステムは、あくまでも言語研究に利用できるツールの一つとして開発したものである。利用にあたっては、各研究者が本システムを利用してデータベースにどのような資料を組み込むのか（ソースデータの選定）が重要である。さらに言えば、研究に対応したデータベースの分類設計も重要である。英語の場合、地域、年代、テキストデータの出版形態などでデータベースを分けて作成し、利用時に組み合せて使用するなどといった利用が望まれる。複数のデータベースから選択、あるいは組み合わせて利用できることも、本システムの特徴の一つである。

さらに重要な点は、データベースに取り込むデータを事前に、緻密に吟味することである。たとえば新聞記事のテキストファイルを例にとると、時系列別にバージョンアップされた記事が、それぞれファイルに記録されていることがある。この場合、特定のセンテンス、語が重複してカウントされるので、統計資料などの取り扱いを慎重にする必要が生じる。いずれにしても、研究者にはソースデータについての理解がシステム利用の前提として求められる。

本稿で論じてきたのは、Bank of English の特に picture の利便性に着目して取り組んだコーパスからの情報抽出システムの設計方針であったが、3 節で述べた通り、Bank of English のシステムそのものは構築者によって提供されたものであり、利用者にはブラックボックスである。これに対して本システムでは、研究目的に適ったデータを研究者自らが準備し、そのデータを基にしてデータベースを構築し、必要な情報を検索・入手できるので、透明性と有用性が高いシステムであるといえる。また、およそ 1 億語のテキストに対して 3 時間程度という比較的短時間でデータベース構築ができることなども、本システムの特徴と言える。今後、更なる機能向上と作業効率の高速化を図る予定である。

---

[1] 本研究は、科学研究費補助金（挑戦的萌芽研究（平成 25 年度～平成 27 年度）、研究代表者：滝沢直宏、課題番号：25580126）の助成を受けている。

[2] 國友は「柔軟性に欠けるきらいがある」について具体例を挙げていないが、第 1 著者（西村）の経験では、たとえばデータ項目に変更が発生すると、関連プログラムすべての修正が必要になり、規模の大きなシステムになるほど柔軟なデータベース変更が困難になるという点がある。

[3] RDBMS とはリレーショナルデータベースを管理するソフトウェアの総称である。

[4] MySQL はオラクル社が商用ライセンスと GPL(General Public License)双方のライセンス形式で提供している。第1著者(西村)に利用経験のある GPL 版の MySQL を選定した。

[5] 識別コードは当該パラグラフのファイル内における一連番号を採用する。fnosno は file no. + segment no.の略。BNC では 1 行のデータをセグメントと称しているのでこれに倣った命名である。

[6] 正規化はリレーショナルデータベースに固有ではなく、階層構造型のデータベース設計でも重視される基礎技術といえる。本節の例示は正規化基準をかなり簡単に述べているので、正規化そのものについては参考図書に挙げた國友(1994: 66ff)や Descartes and Bunce (2000: 144)などを参考にされたい。

[7] 座席予約システムにおけるダブルブッキングを想像するとわかりやすい。

[8] システム障害などに備えて、データベースのバックアップ体制を整えることは必要である。

[9] 表 1 内の度数は、表 3 に示すコーパスデータベースビルダーbuild_db_bncx で作成したデータベース db_bncx 内にあるテーブル k_lemma をもとに作成した。

[10] 注 8 と同じ方法で作成した。

[11] この評価により、実際にはプログラムを作成しなかった。

[12] k は単独レマ、u は複数レマで構成されるテーブルである。i は $tkn\_k_{1\text{-}500}$、$tkn\_u_{1\text{-}400}$。この基準は BNC のレマ度数分布をもとに設定した（表 1、2 を参照）。

[13] BNC のファイル容量は約 4,571MB、このファイルから作成したデータベース db_bncx の容量は 29,901MB である。

[14] BNC の以前のバージョン World Edition では lemma が付加されていない。

[15] TreeTagger に関しては次の URL を参照。
www.cis.uni-muenchen .de/~schmid/tools/TreeTagger/

[16] BNC をソースにデータベースを構築した処理時間は約 2 時間 40 分である。おおむね 1 億語

のデータベース作成が 3 時間程度で可能であると見込んでいる。
[17] L5～R5 については 6.7 節を参照。
[18] 記号類も含むため、「要素」という用語を使っている。

参考文献

Descartes, Alligator and Tim Bunce (2000) *Programming the Perl DBI.* Sebastopol, CA: O'Reilly Media Inc.

情報システムの政府調達に係る SLA 導入研究会 (2004)『情報システムに係る政府調達への SLA 導入ガイドライン』独立行政法人情報処理推進機構. [http://www.meti.go.jp/policy/it_policy/tyoutatu/sla-guideline.pdf]

國友義久 (1994)『情報システムの分析・設計』日科技連出版社.

齊藤俊雄・中村純作・赤野一郎 (2005)『英語コーパス言語学:基礎と実践』改訂新版. 研究社.

Schwartz, Baron, Peter Zaitsev, Vadim Tkachenko, Jeremy D. Zawodny, Arjen Lentz and Derek J. Balling (2008) *High Performance MySQL Second Edition.* Sebastopol, CA: O'Reilly Media Inc.

滝沢直宏 (1998)「Cobuild*Direct* を利用したデータ収集―英語語法・文法研究のための基礎作業―」『言語と人間』2: 151-169. 朝日出版社.

八木克正・井上亜依 (2013)『英語定型表現研究―歴史・方法・実践―』開拓社.