# テキストとアノテーションの汎用同時検索システム

狩野 芳伸（科学技術振興機構 さきがけ）† 増田 勝也（東京大学 大学総合教育研究センター）

# A Generic Searching System for Text and Annotations

Yoshinobu Kano (PRESTO, Japan Science and Technology Agency)　Katsuya Masuda (Center for Research and Development of Higher Education, the University of Tokyo)

テキストデータの検索は、商用・無償ともに様々なシステムが提供されており、広く利用される一般的な技術になりつつある。一方で、テキストデータに付与されたアノテーションの検索については、理論・実装いずれも整備されているとはいいがたい状況であり、標準的なツールも存在しない。しかし、複雑な構造のアノテーションから研究者が興味あるアノテーションを抜き出すことは、自然言語処理など工学的な分野だけでなく言語学など事例に基づく研究においても重要な作業である。また、異なる研究者が開発したアノテーションを同時に検索できるようにすることは、アノテーションの利用において大きな意味がある。本稿では、BCCWJ コアデータを主な対象として、テキストとアノテーションを同時に検索するためのクエリ設計と、そのクエリで検索を行う汎用検索システムの実装について述べる。

## １．はじめに

本研究の目的は、将来の新規アノテーション種へも対応できるよう、検索の下位レイヤを汎用のシステムを設計し実装することにある。すなわち、ユーザビリティの向上や GUI の開発などは将来の課題とし、本研究では拡張性のある汎用システムを構築した。

前述のように、テキスト一般の検索システムについては高性能かつ安定したものが長年にわたりオープンソースで公開され、非常に多くのユーザに利用されている。この部分を新たに構築しても、性能面でも安定性でも勝るものを作成するのは現実的に無理である。そのため、テキスト検索に関わる部分はデファクトスタンダードのオープンソースライブラリである Apache Solr/Lucene[1]を内部的に利用することとした。執筆時の最新メジャーバージョンである Solr 4 では、速度向上とスケーラビリティ向上の面で大幅な機能追加がなされており、すでに多くの企業において超大規模のテキスト検索サービスを構築するために利用されている。

アノテーションの検索システムについては、知る限り標準といえるものが存在しない。本研究では内部的に Solr を用いつつ、アノテーションとテキストの検索ができるシステムを新たに構築した。アノテーション検索に最も近いと思われる技術は XML の検索であるが、整形式の XML 文書ではタグの交差が許されないため、アノテーション一般の検索をすることはできない。我々の設計では、アノテーションの検索クエリについて領域代数（Region Algebra）を基本とし、アノテーション一般の検索に対応する汎用性を担保した。

## ２．背景

### ２.１ Apache Solr

Apache Solr は、Java で実装されたオープンソースの全文検索ライブラリである。Solr のコア部分は Apache Lucene と呼ばれており、Solr はおおむね Lucene にさまざまな付加機能を追加したものになっている。Solr の機能は多岐にわたり、バージョンアップの度に新機能が追加され非常に活発に開発が続けられている。Solr 4 では SolrCloud 機能が追加

† kano@nii.ac.jp

[1] http://lucene.apache.org/solr/

され、複数台構成のサーバ群を一つの検索システムとして運用するための機能と、それを容易に管理できる各種ユーテリティが追加された。

Solr の特徴は、多くのユーザを持つことによる安定性と関連情報量の多さに加え、非常に高速な検索を実現していることが挙げられる。これは同時にスケーラビリティにも優れていることを意味する。Solr ウェブサイトのデータ [2]によると、たとえば電子図書館 HathiTrust でのサービスにおいて、500 万冊の書籍を全文検索させている [3]。100 万冊の場合のベンチマークを以下に引用する。テキストデータサイズは 666GB とあるので、100 万冊は日本語の場合 333G 文字程度、平均単語長が 3 文字程度とすると 100G（一千億）単語程度に相当すると思われる。インデックスサイズはデータの 35%程度である。このベンチマークはやや古いため、サーバは一台、サーバの搭載メモリは 4GB-8GB 程度である。ほとんどのクエリに 1 秒以内で返答している。現在においては、SSD の利用とマルチコアサーバにより、さらに数倍以上の高速化が安価に可能と考えられる。

### ２.２ 領域代数

領域代数(Clarke et al.,1995, Jaakkola et al. 1999, Masuda et al. 2009)とは領域集合間の関係性を記述する演算の集合である。領域は連続するテキスト列として定義され、開始・終了位置の組で表現することができる。各演算は領域集合を引数としてとり、演算結果としてまた領域集合を返す。本研究においては、タグで囲われたテキスト範囲を領域として考えることで、領域代数表現をクエリ、演算処理を検索実行処理として、領域代数をアノテーションの付与されたテキストに対する検索として利用する。領域代数には標準的な演算集合といえるものは存在しないが、我々は以下で説明する演算集合を採用する。説明をわかりやすくするため、XML/HTML 風のタグ記述を用いて説明する。また以下では領域集合 A,B に対する演算を考える。A,B はその名前を持つタグ(<A>,<B>)によるアノテーション領域の集合であり、a∈A, b∈B は単一のアノテーション領域を指す。

#### ２.２.１ 包含演算（contain）

包含演算子は、領域 a が領域 b を包含することを表す。つまり<A>…<B>…</B>…</A>という関係である。演算結果は上記の条件を満たす領域 a の集合となる。

#### ２.２.２ 被包含演算（contained in）

被包含演算子は、領域 b が領域 a を包含することを表す。つまり<B>…<A>…</A>…</B>という関係である。演算結果は上記条件を満たす領域 a の集合となる。包含演算子とは演算結果が包含する側の領域か包含される側の領域であるか、という点で異なる。

#### ２.２.３ 順序演算（follows）

順序演算子は、領域 b が領域 a より後に来る領域を返す。つまり<A>…</A> … <B>…</B>という関係である。ただし領域 a と 領域 b が交差する場合は除く。領域としては、a で始まり b で終わる領域となる。

#### ２.２.４ AND 演算（AND）

AND 演算子は、領域 a と領域 b が同時に出現する領域を返す。領域としては、a で始まり b で終わる領域又はその逆の領域を指す。主に包含演算子と組み合わせることで、”A,B の両方を含む C”のような記述が可能である。

#### ２.２.５ OR 演算（OR）

OR 演算子は、領域 a または領域 b のいずれかの領域を返す。領域集合としては領域集合 A と領域集合 B の和集合となる。

#### ２.２.６ 重なり演算（overlaps）

重なり演算子は、領域 a と領域 b が部分的に重なっている領域を返す。つまり<A>…<B>…</A>…</B>という関係である。領域集合としては重なっている領域、先の例での<B> … </A> の領域の集合となる。

---

[2] http://wiki.apache.org/solr/SolrPerformanceData

[3] http://www.hathitrust.org/technical_reports/Large-Scale-Search.pdf

### ２.３ 大規模日本語均衡コーパス BCCWJ

国立国語研究所で開発された BCCWJ(前川, 2009)は日本語において最大規模の均衡コーパスである。テキストは新聞・小説・ブログなどさまざまなジャンルのものからランダムに選択されたものが収録されている。

BCCWJ の一部データに対ついては、テキストに対しアノテーション情報が付与されている。どのアノテーションがどの部分に付与されているかはアノテーションによって異なり、重複する部分もあればしない部分もある。

BCCWJ の利用には国語研究所との間で契約が必要で、契約者にはデータを収録した DVD が配布されている。現在配布されている DVD には、テキスト情報に加え形態素レベルの情報が収録されている。形態素は短単位（SUW）と長単位（LUW）が別個に付与されている。これらの情報は、基本的に XML 形式で記述されている。本稿では BCCWJ の詳細については記述しない。

それ以外のアノテーションは、DVD には現在収録されていない。本研究では国語研が別途配布している係り受けアノテーション DepPara2[4]・奈良科学技術先端大学院大学を中心に開発された述語項構造(小町ほか, 2011)・慶応大学の開発した日本語フレーム構造(小原ほか., 2011)・岡山大学の開発した動詞項構造(竹内ほか, 2012)を用いた。

### ２.４ 国際標準 UIMA フレームワーク

UIMA (Unstructured Information Management Architecture)(Ferrucci et al., 2006)は様々な企業・研究機関で利用されている相互運用性のための枠組みである。UIMA はメタデータとそのための API を提供しており、実装は Apache　UIMA としてオープンソースで提供されている。最近では、テレビ番組のクイズ王に勝利した IBM の Watson システム(Ferrucci, 2012)でも用いられた。

UIMA の実行単位はコンポーネントと呼ばれ、コンポーネントを組み合わせて実行可能な UIMA ワークフローを作成する。おおむね、コンポーネントはいわゆるツールに相当し、ワークフローはアプリケーションに相当する。実行時のデータ構造は CAS と呼ばれる汎用構造に統一されている。XML のようなインライン形式と異なり、CAS ではスタンドオフ形式を前提としている。スタンドオフ形式ではテキストとアノテーションが分離され、アノテーションはテキスト内のオフセット位置を参照することでテキストに紐づけされる。CAS は概念的な構造であり、実行時は Java ヒープ内にオンメモリで格納されるが、XMI(XML Metadata Interchange)などファイル形式で保存することもできる。

UIMA は他に、データ型を階層的に定義する type system を記述する XML 形式やコンポーネントのメタデータ記述の XML 形式などを提供している。UIMA のアノテーションは type system により定義済みのデータ型で型付けされなければならない。UIMA 自体は整数や文字列といった基本的な型しか提供しないため、開発者が必要に応じてアノテーションの型階層を定義する必要がある。型階層は木構造で、型には素性と呼ぶ属性値を定義することができる。また、コンポーネントそのものも基本的に個々の開発者が作成して提供することになっており、様々な研究グループ・企業からコンポーネントが提供されている。

ウェブサービスの形式としては、ウェブサービスコンテナである Apache ActiveMQ 上で展開される UIMA-AS サービスが用意されている。

### ２.５ 統合自動自然言語処理システム Kachako

Kachako は UIMA 準拠のプラットフォームと互換ツールキット（UIMA コンポーネント群）からなる、統合全自動言語処理システムである(狩野, 2012)(Kano, 2012)。Kachako プラットフォームは、ツールの選択・組合せ・並列分散展開実行・視覚化・評価までを徹底的に自動化している。ツールキットでは統一 type system を定義した上で、その type system に互換な UIMA コンポーネント群を構築して、プラットフォームと互換ツール群をポータブルな形で配布し、ユーザの指定した任意の計算資源上でインストールから大規模処理ま

[4] https://sites.google.com/site/masayua/bccwjdep

で自動実行することを可能にしている。

Kachako の特徴の一つは、自動ワークフロー生成機能である。Kachako はコンポーネント群から可能なワークフローを自動計算するためにコンポーネントの入出力データ型情報を用いる。そのためデータ型の定義は自動計算が可能なように設計すると同時に、コンポーネントの入出力条件を過不足なく表現できるようにする必要がある。

我々は本研究に先行して、BCCWJ の各種アノテーションを Kachako および UIMA 互換となるよう変換するコンポーネントを実装した。本研究では入力を UIMA 形式とすることで汎用設計とし、入力データには BCCWJ データから変換した入力を用いた。

## ３．クエリの設計

テキストとアノテーションの統合検索を行う場合、大きく分けて三つの検索対象が考えられる。ひとつは、テキストそのものや、単一のアノテーションといったリテラル値である。二つ目は、文書内のアノテーション間の位置関係である。三つめは、アノテーション間の明示的な参照関係である。ありうる検索パターンをカバーした汎用的な検索のためには、これら三つの要素を複合的に組み合わせた検索クエリが必要である。

### ３.１ リテラルのクエリ

テキストのリテラル値は、二重引用符でくくった文字列で表記する。たとえば

```
"word"
```

のようになる。

アノテーションのリテラルについては、角括弧でくくった中に、一般的な XML/HTML タグと同様の表現でアノテーション名および属性名・属性値のペアを指定する。属性名と属性値は等号で接続し、属性値は二重引用符で囲う。アノテーション名と属性名は空白で区切る。たとえば

```
[annotation attr="value"]
```

のようになる。属性値を指定するかどうかは任意であり、指定しなくともよい。指定しない場合は、そのアノテーション名を持つ全てのアノテーションを表す。

### ３.２ 領域代数によるアノテーション間の位置関係を指定するクエリ

領域代数のクエリは、上記リテラルのクエリと領域代数の演算子を用いてポーランド記法で記述する。各演算は二項演算であり、表現としては全体が丸括弧でくくられ、最初に演算子、その後演算子のとる項が二つ並ぶ。記号と項は空白で区切る。演算子のとる項は、リテラル値または別のクエリの埋め込みである。つまり、ひとつのクエリは、複数の演算子やリテラルが複合的に組み合わさった表現であり、演算子が中間ノードでリテラルが葉ノードの二分木として表現可能である。

前述の領域代数の各演算に対する演算子として以下の記号を使用する。

包含:　`(> A B)`
被包含:　`(< A B)`
順序:　`(- A B)`
AND:　`(& A B)`
OR:　`(| A B)`
重なり:　`(@ A B)`

### ３.３ アノテーション間の参照を指定するクエリ

#### ３.３.１ 変数

変数は、その変数の位置に任意の値が入ることを表現する。変数は'$'で始まる文字列で表記し、たとえば

```
[annotation attr="$1"]
```

は、annotation の attr 属性に任意の値が入ることを表現する。一クエリ中の複数箇所に同一名の変数が出現する場合は、それらの箇所には同一の値が入るとする。

### ３.３.２ リンクの表現

アノテーションの属性によっては、ほかのアノテーションを参照するリンクを持つことがある。このような場合は、上記の変数とアノテーション属性を用いてリンクを表現する。たとえば

```
(& [annotation attr="$1"] [annotation2 id=”$1”])
```

という表現は、annotation の attr 属性と annotation2 の id 属性が同じ値を持つことを表現する。すなわち annotation の attr 属性が annotation2 の id 属性を参照しているような領域を検索せよ、という意味になる。なお、このようなリンク表現を使用する際には、参照関係があらかじめアノテーションによって記述されている必要がある。

### ３.３.３ 領域を持たないアノテーションへのリンク

リンク先のアノテーションが領域を持たない（テキストの中の位置に紐づけられていない）ものに関しては、リンク先の属性名をさらにつづけて記述することで、領域を持たないアノテーションを埋め込む形で記述する。属性名はコロンで区切る。たとえば

```
[annotation attr1:attr2="val2"]
```

という場合、annotation の属性 attr1 の参照先のアノテーションの attr2 属性の値が val2 の場合、という意味になる。

## ４．検索システムの設計と実装

検索システムはすべて、Java 言語で実装した。外部ライブラリもすべて Java による実装のため、事実上ほとんどあらゆるサーバ、PC 上で稼働する。

### ４.１ リテラルの検索

リテラルの検索には、Solr を用いる。インデックス手法は標準的な転置インデックスまたは n-gram インデックスを想定する。転置インデックスはより高速な検索が必要な場合、n-gram インデックスは漏れのない検索が必要な場合に用いる。Solr のサポートするインデックス手法は多様であり、API に互換性があれば他の手法も利用可能である。

テキストのリテラル値インデックス作成は、一般のテキスト検索と変わりがない。

アノテーションのリテラル値インデックスは、非トークンを表す特殊なフィールドに保持し、インデックス自体はテキストのリテラル値と統合して検索するがトークン分割処理は行わない。アノテーションをインデックスする際は、属性値ごとにインデックスを作成する。たとえば

```
<annotation attr1="value" attr2="value2"> … </annotation>
```

というアノテーションに対しては

```
annotation:attr1="value1"
annotation:attr2="value2"
```

の二つがインデックスに追加される。これにより、インデックスの肥大を抑えつつ、インデックス検索のみで効率的な検索が可能になる。

### ４.２ 領域代数クエリの検索

領域代数クエリの検索は、クエリの構成する演算子二分木の各演算子ノードを一つ一つ処理することで行う。演算子の処理は、条件に合致するインデックスエントリを Solr 経由で取得することで行う。Solr の検索結果からはヒットしたアノテーションの begin/end 値が直接取得できるので、この値によるアノテーション間の位置関係についての条件の一致を行うことで高速に検索する。

この仕組みでは、演算子ごとに毎回 Solr のインデックス検索が実行され、演算子が複合的に埋め込まれている場合は取得した候補リスト中の各アノテーションについてさらにインデックス検索を実行するということが入れ子で繰り返される。

### ４.３ 変数の検索

変数の検索も、領域代数クエリの検索と同様であるが、まず変数に入りうる属性値を別途

検索して列挙する。そしてそれらの値を対応する変数に代入したクエリ表現を生成し、領域代数クエリの検索を行う。

## ５．おわりに

本研究では、BCCWJ コアデータおよびコアデータへのアノテーションを検索対象に想定して、汎用のテキスト・アノテーション統合検索システムを構築した。今後の課題としては、より大規模なデータでのスケーラビリティのテストが挙げられる。テキストのみの検索であれば Solr は十分なスケーラビリティを持つと考えらえるが、アノテーションの同時検索の場合データ量が大幅に増えるうえ検索時間が長くなるため、検証が必要である。

また、領域代数を基礎とするクエリは汎用ではあるが、文科系を含め専門外の研究者にとっては利用のハードルが高いことが想定される。より容易にクエリを入力できるようなクエリ生成支援機能や GUI の実装、検索結果表示方法の改善を検.討していきたい。

## 謝　辞

本研究の一部は、国立国語研究所共同研究プロジェクト「コーパスアノテーションの基礎研究」（プロジェクトリーダー：前川喜久雄）および科学技術振興機構さきがけ「情報環境と人」領域「解析過程と応用を重視した再利用が容易な言語処理の実現」（研究代表者：狩野芳伸）による補助を得ています。

## 文　献

C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. The computer Journal, 38(1):43–56, 1995

J. Jaakkola and P. Kilpel¨ainen. Nested Text-Region algebra. Technical Report C-1999-2, University of Helsinki, 1999.

Katsuya Masuda and Jun'ichi Tsujii. Tag-Annotated Text Search Using Extended Region Algebra. The IEICE Transactions on Information and Systems, Special Section on "Natural Language Processing and its Applications," Vol.E92-D,No.12, pp.2369-2377, 2009.

前川喜久雄. (2009). 代表性を有する大規模日本語書き言葉コーパスの構築. 人工知能学会誌, *24*(5), pp.616–622.

小原京子, 加藤淳也, 斎藤博昭. (2011). 日本語フレームネットにおける BCCWJ への意味アノテーション. *日本語コーパス平成 22 年度公開ワークショップ* (pp. 513–518).

小町守, 飯田龍. (2011). BCCWJ に対する述語項構造と照応関係のアノテーション. *日本語コーパス平成 22 年度公開ワークショップ* (pp. 325–330).

竹内孔一, 竹内奈央, & 石原靖弘. (2012). 述語の分析に基づく文書解析の考察. *IPSJ SIG Notes* (Vol. 2012, pp. 1–7). Information Processing Society of Japan (IPSJ).

Ferrucci, D. (2012). Introduction to "This is Watson". *IBM Journal of Research and Development*, *56*(3.4), 1:1–1:15. doi:10.1147/JRD.2012.2184356

Ferrucci, D., Lally, A., Gruhl, D., Epstein, E., Schor, M., Murdock, J. W., Frenkiel, A., et al. (2006). Towards an Interoperability Standard for Text and Multi-Modal Analytics. IBM Research Report.

狩野芳伸. (2012). Kachako: 誰でも使える全自動自然言語処理プラットホーム. *2012 年度人工知能学会全国大会（第 26 回）*.

Yoshinobu Kano. (2012) Kachako: a Hybrid-Cloud Unstructured Information Platform for Full Automation of Service Composition, Scalable Deployment and Evaluation. *In the 1st International Workshop on Analytics Services on the Cloud (ASC), the 10th International Conference on Services Oriented Computing (ICSOC 2012)*. Shanghai, China, November 12nd 2012.