# GCL問い合わせ代数による構造化テキストのランク検索

増田 勝也† 二宮 崇‡* 辻井 潤一‡*
†東京大学理学部情報科学科 ‡CREST, 科学技術振興事業団
*東京大学大学院情報理工学系研究科コンピュータ科学専攻
{kmasuda,ninomi,tsujii}@is.s.u-tokyo.ac.jp

## 1 はじめに

本論文では、タグにより半構造化されたテキストに対し適用可能なGCL問い合わせ代数 [1] に、ランキング技術 [2] を組み込むことにより、Web上のテキストに対して、拡張性、頑健性、効率性を達成する実用的な近接構造検索アルゴリズムを提案する。

近接構造検索を実現する既存の枠組みの一つであるGCL問い合わせ代数とそのアルゴリズムは、テキスト範囲の包含関係を利用した近接構造検索を実現する。しかしながらGCL問い合わせアルゴリズムは比較的整理された小規模なテキストを対象としたアルゴリズムであり、Webに代表される記述に揺れのある大規模テキストに適用することは想定されていない。まず、GCL問い合わせアルゴリズムはクエリと完全にマッチするテキスト範囲のみ出力するが、Web上のテキストは常にクエリで示される構造を完全に満たす表現で記述されているとは限らず、頑健性の問題が生じる。また、Web上の検索で求められている検索結果はよりクエリとの関連性が高いテキスト範囲であるが、この手法ではクエリと完全にマッチするテキスト範囲は関連度が低いものも含めてすべて計算され出力されるため、単純にGCL問い合わせアルゴリズムの出力に対してスコア付けをする手法では効率面の問題が生じる。

本手法ではこのGCL問い合わせ代数にランキング技術を組み込むことによって、ランク付き近接構造検索を実現する。スコア計算の際に単語と構造の両方に対してスコアを付与することにより、クエリと完全にマッチしないテキスト範囲でもクエリとの関連度が高ければ出力される。また関連度の高いテキスト範囲のみを計算することにより、大規模テキストに対しても検索を効率的に行うことができる。

## 2 背景

### 2.1 Generalized Concordance List (GCL) 問い合わせ代数

GCL問い合わせ代数 [1] はタグによってマークアップされたテキストに対する近接構造検索のクエリを表現する問い合わせ代数である。テキスト中の開始タグと終了タグに囲まれた範囲は互いにオーバーラップしていてもかまわない。検索の結果は**extent**と呼ばれるテキスト中の範囲を示す整数のペアの集合とする。また、extent間の

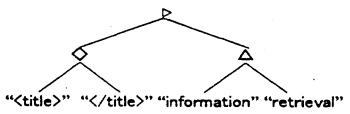


図1: クエリ ( "⟨title⟩" ◇ "⟨/title⟩" ) ▷ ( "information" △ "retrieval") を表す木構造

二項関係としてnestが定義されている。extent $a = (p, q)$ および $b = (p', q')$ に対し、$p' \leq p \leq q \leq q'$ が成り立つとき、aはbにnestされるといい、$a \sqsubset b$ で表す。extentの集合GCLは、各要素がお互いにnestしないextentの集合であり、extentの集合 $S$ に対するGCL $G(S)$ は次のように定義される。

$$G(S) = \{a | a \in S \wedge \nexists b \in S.(b \neq a \wedge b \sqsubset a)\}$$

extentの集合 $A$, $B$ に対して以下の7種類の二項演算子が定義される。

| | |
|---|---|
| Containing | $A \rhd B = G(\{a \mid a \in A \wedge \exists b \in B.(b \sqsubset a)\})$ |
| Contained In | $A \lhd B = G(\{a \mid a \in A \wedge \exists b \in B.(a \sqsubset b)\})$ |
| Not Containing | $A \not\rhd B = G(\{a \mid a \in A \wedge \nexists b \in B.(b \sqsubset a)\})$ |
| Not Contained In | $A \not\lhd B = G(\{a \mid a \in A \wedge \nexists b \in B.(a \sqsubset b)\})$ |
| Both Of | $A \triangle B = G(\{c \mid c \sqsubset Z \wedge \exists a \in A.\exists b \in B.(a \sqsubset c \wedge b \sqsubset c)\})$ |
| One Of | $A \triangledown B = G(\{c \mid c \sqsubset Z \wedge \exists a \in A.\exists b \in B.(a \sqsubset c \vee b \sqsubset c)\})$ |
| Followed by | $A \diamond B = G(\{c \mid c = (p, q') \text{ where } \exists (p, q) \in A.\exists (p', q') \in B.(q < p')\})$ |

$Z$ は最大のextent$(-\infty, \infty)$ を表す。$\rhd$, $\lhd$, $\not\rhd$, $\not\lhd$ の4つの演算子によってテキスト上の構造の包含関係を表すことができる。

これらの演算子で表されるクエリは木構造で表すことができる。図1はクエリ "( "⟨title⟩" ◇ "⟨/title⟩" ) ▷ ( "information" △ "retrieval")" を表す木構造を示す。

GCL問い合わせ代数による検索の例を図2に示す。この例では (("⟨html⟩" ◇ "⟨/html⟩") ▷ ("information" △ "retrieval")) で "information" 及び "retrieval" を含む "⟨html⟩" ◇ "⟨/html⟩" で囲まれた範囲 (Webページ) を表し、("⟨title⟩" ◇ "⟨/title⟩") で "⟨title⟩" と "⟨/title⟩" で囲まれた範囲 (タイトル) を表す。クエリ全体でそのよ

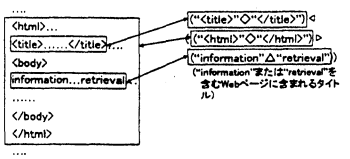


図 2: (“⟨title⟩” ◇ “⟨/title⟩”) ◁ ((“⟨html⟩” ◇ “⟨/html⟩”) ▷ (“information” △ “retrieval”)) に対する検索の例

うな Web ページに含まれるタイトルを表し、タイトルの部分が出力される。

## 2.2 ランク検索

Web 上のキーワード検索に代表されるランク付き近接検索 [2] は単語の集合をクエリとして受け取り、クエリに対する各ドキュメントの関連度をスコアで表し、スコアの順にソートされたドキュメントをユーザーに提示する。ドキュメント $d$ のクエリ $q$ に対するスコアは

$$S(d,q) = \frac{1}{W_d W_q} \sum_{t \in q \cap D_d} w_{q,t} \cdot w_{d,t}$$

で表される ($w_{q,t}$: 問い合わせ $q$ に対する単語 $t$ の重み、$w_{d,t}$: ドキュメント $d$ における単語 $t$ の重み、$W_d = \sqrt{\sum w_{d,t}}$、$W_q = \sqrt{\sum w_{q,t}}$、$D_d$: ドキュメント $d$ に現れる単語の集合)。

$w_{q,t}$ 及び $w_{d,t}$ の定義には TF×IDF 法 [3] による定義が一般的によく用いられる。TF×IDF 法は、

- ひとつのドキュメントに多く含まれる単語は重要である (Term Frequency)
- 現れるドキュメントが少ない単語は重要である (Inversed Document Frequency)

という 2 つの基準により各単語の重みを決める。ドキュメント $d$ における単語 $t$ の出現回数を $f_{d,t}$、$t$ が出現するドキュメントの数を $f_t$ とすると、単語 $t$ のドキュメント $d$ における重みは、最も単純なものとしては $\frac{f_{d,t}}{f_t}$、他の例としては $(1+\log_e f_{d,t}) \cdot \log_e(1+\frac{N}{f_t})$ で表される ($N$:ドキュメントの数)。

すべてのドキュメントに対してスコアを計算することは高コストであるため、1) インデックス付けと 2) スコアの近似計算 [2] を行う。

1) インデックス付けは検索システムに登録された単語集合 (=辞書) に対して行われる。インデックスは辞書の要素である各単語 $t$ に対して得られるリスト構造であり、各リストの要素は $t$ を含むドキュメント $d$ へポインタ $p_d$ とドキュメント $d$ での $t$ の重み $w_{d,t}$ のペア $(p_d, w_{d,t})$ である。

2) スコアの近似計算は以下の方法で行う。クエリが与えられた後、クエリに現れる単語のインデックス中で最も TF×IDF 値が高いペア $(p_d, w_{d,t})$ に対して、ドキュメント $d$ に対するアキュムレータ (近似計算されたスコアを保持する場所) に $w_{d,t}$ を加える。ドキュメント $d$ に対するアキュムレータが無ければ新たに作る。スコアを計算するドキュメントの数は、アキュムレータの最大数 $T$ をあらかじめ決めておき、アキュムレータの数が $T$ に到達したら、新たにアキュムレータを作らないことで制限する。

# 3 アルゴリズム

GCL 問い合わせ代数による検索では、一般の検索におけるドキュメントという単位があらかじめ決められていない。本手法では「**最大範囲クエリ**とマッチする extent」を一般の検索におけるドキュメントに対応する単位とし、一般のランク検索で用いられるアルゴリズムを最大範囲クエリに対して適用する。クエリで求められている範囲を示す**出力範囲クエリ**を求め、これら 2 つのクエリとマッチする extent のペアに対しスコアを計算する。

**出力範囲クエリ** $Q_o$ :与えられたクエリの中で結果として出力される extent を表す部分クエリ。

**最大範囲クエリ** $Q_m$ :与えられたクエリの中で最も広い extent を表す部分クエリ。

クエリが (“⟨title⟩” ◇ “⟨/title⟩”) ◁ ((“⟨html⟩” ◇ “⟨/html⟩”) ▷ (“information” △ “retrieval”)) の場合 (“information” または “retrieval” を含む Web ページに含まれるタイトル)、出力範囲クエリは (“⟨title⟩” ◇ “⟨/title⟩”) であり、最大範囲クエリは (“⟨html⟩” ◇ “⟨/html⟩”) となる。

すべての extent のペアに対してスコアを計算することは高コストであるので、一般のランク検索と同様にクエリとの関連性の高い extent のペアを決められた数 $T$ 個選び出し、それらに対してのみスコア計算を行う。$T$ の値を制限することにより、高速に計算を行うことができる。さらに関連性の近似計算を高速に行うためにあらかじめ**インデックスリスト**として各 extent に対する各単語のスコアを計算する。しかし、テキスト上のすべての extent に対してのスコアを計算しておくことは事実上不可能であるので、最大範囲クエリはタグで囲まれた形 ⟨m⟩ ◇ ⟨/m⟩ と限定する。

クエリが与えられる前に、テキストが与えられた時点で**インデックス付け**が行われる。インデックス付けではテキストから検索に必要な**インデックスリスト**の計算を行う。クエリが与えられた後のアルゴリズムの流れは以下のようになる。1) **クエリ計算**ではクエリを入力として受け取り、後の計算で使用する出力範囲クエリおよび最大範囲クエリを計算する。2) **候補選択**ではクエリ計算で求められた最大範囲クエリとマッチする extent のうち、クエリとの関連性の高い extent を選び、その extent に含まれる出力範囲クエリとマッチする extent とのペアのリストを作る。3) **スコア計算及びランキング**では、候補選択で選ばれた extent のペアに対してスコア計算を行い、スコア順のランキングリストを作り、出力する。

本手法での TF×IDF 値は、一般のランク検索で使われる TF×IDF 値の「ドキュメント」に当たる部分を「⟨m⟩

```
function OQuery(Q): query;
  begin
    n := root(Q);
    while contain_op(n) do
      n := leftchild(n);
    q := subtree_query(n);
    remove_contain(q);
    return q;
  end

root: クエリの根ノードを返す
contain_op: そのノードの演算子が ▷, ◁, ⋪, ⋫の
  いずれかならば真を返す
leftchild: 左の子ノードを返す
subtree: そのノードを根ノードとする部分木が表すクエリを返す
remove_contain: クエリ中に含まれる ▷, ◁, ⋪, ⋫の
  ノードとその右の子を取り除く
```

図 3: 出力範囲クエリを求めるアルゴリズム

```
function MQuery(Q): query;
  begin
    n := root(Q);
    while contain_op(n) do
      if containing_op(n) then
        n := leftchild(n)
      else
        n := rightchild(n)
    q := subtree_query(n);
    remove_contain_max(q);
    return q;
  end

containing_op: そのノードの演算子が ▷ または ⋪ならば真を返す
rightchild: 右の子ノードを返す
remove_contain_max: クエリ中に含まれる ▷, ⋪のノードと
  その右の子、及び ◁, ⋫のノードとその左の子を取り除く
```

図 4: 最大範囲クエリを求めるアルゴリズム

◇ ⟨/m⟩ とマッチする extent」と置き換えたものである。すなわち、以下のように定義する。

$$S_{t,E} = (1 + \log_e g_{E,t}) \cdot \log_e (1 + \frac{N_m}{f_{m,t}})$$

ただし、extent $E$ は ⟨m⟩ ◇ ⟨/m⟩ とマッチするとする。$N_m$ はデータベース全体での ⟨m⟩ ◇ ⟨/m⟩ とマッチする extent の数、$f_{m,t}$ は ⟨m⟩ ◇ ⟨/m⟩ とマッチし、かつ単語 $t$ を含む extent の数、$g_{E,t}$ は extent $E$ 中に含まれる単語 $t$ の数とする。

## 3.1 インデックス付け

インデックス付けはクエリが与えられる前に行われ、インデックスリスト $I_{t,m}$ を各単語 $t$ 及びタグ $m$ に対して作成する。インデックスリストの要素は extent とその extent に対する単語 $t$ の TF×IDF 値のペアである。TF×IDF 値を計算するためには3個の値 ($N_m$, $f_{m,t}$, $g_{E,t}$) が必要となる。そこでテキストを読んでいく際に、終了タグを読んだ後それに対応する開始タグからその終了タグの範囲に含まれる単語に応じてそれらの値を変化させる。

## 3.2 クエリ計算

クエリ $Q$ が与えられた後、後の計算で使用する出力範囲クエリと最大範囲クエリの二つの部分クエリを求める。図 3、4 はそれらのクエリを求めるアルゴリズムを示す。

出力範囲クエリに包含関係の条件を含めないために、▷、◁、⋪、⋫が含まれている場合、その演算子のノードと、▷ または ⋪ の場合は右の子、◁ または ⋫ の場合は左の子を取り除く。たとえば、出力が (("⟨a⟩" ◇ "⟨/a⟩") ▷ "search") △ (("⟨b⟩" ◇ "⟨/b⟩") ▷ "algorithm") の場合、出力範囲クエリは ("⟨a⟩" ◇ "⟨/a⟩") △ ("⟨b⟩" ◇ "⟨/b⟩") となる。

```
function SELECT(T,Q,Q_o,I): list;
  begin
    WList := word(Q);
    foreach t ∈ WList
      foreach (E, S_{t,E}) ∈ I_{t,m}
        putin(MList,(E, S_{t,E}));
    sort(MList);
    while |MList| ≠ 0 and |PList| < T do
      begin
        (E, S_{t,E}) := highest(MList);
        EList := contained(E);
        foreach E_i ∈ EList
          putin(PList,(E_i, E));
        remove(MList,(E, S_{t,E}));
      end
    return PList;
  end

word: クエリ中の単語のリストを返す
putin: リストに要素を加える
sort: リストの要素をスコアの高い順に並べる
hightest: リストの中でスコアの最も高いペアを取り出す
contained: extent に含まれる Q_o を満たす extent のリストを返す
remove: リストから要素を取り除く
```

図 5: 候補選択のアルゴリズム

## 3.3 候補選択

テキスト上の $Q_m$ とマッチする extent の集合からクエリとの関連度の高い extent を選び出し、その中に含まれる $Q_o$ とマッチする extent とのペアを候補として取り出す。関連度は、クエリ中に現れる単語の $Q_m$ とマッチする extent での TF×IDF 値で表す。

候補を選び出すアルゴリズムは図 5 で表される。入力として選び出す候補の数 $T$、クエリ $Q$、出力範囲クエリ $Q_o$、各単語のインデックスリスト $I$ をとり、候補のペアのリスト $PList$ を出力する。このリストの各ペアに対してスコア計算を行う。

## 3.4 スコア計算

スコア計算はクエリを木で表した時の各ノードにスコアを与えていき、根ノードのスコアを extent のペア $(E_o, E_m)$ のスコアとする。extent のペア $(E_o, E_m)$ のクエリ $Q$ に対するスコアを $SE_{E_o,E_m,Q}$ とする。

葉ノード (単語 $t$) のスコアは先に定義した本手法での TF×IDF 値を利用して、$S_{t,E_m}$ で定める。

また葉ノード以外のノードのスコアは、以下のように

定義される (そのノードの演算子を $op$、そのノードを根とする部分木が表すクエリを $Q(=Q_1 op Q_2)$ とする)。
演算子 $op$ が $\triangleright$、$\triangleleft$、$\triangle$、$\triangledown$、$\diamond$ の場合

$$SE_{E_o,E_m,Q_1 op Q_2} = \begin{cases} SE_{E_o,E_m,Q_1} + SE_{E_o,E_m,Q_2} \\ \quad L_{E_o,E_m,Q}\text{の要素が無い場合} \\ (SE_{E_o,E_m,Q_1} + SE_{E_o,E_m,Q_2})^2 \\ \quad L_{E_o,E_m,Q}\text{の要素があり、和の値が正の場合} \\ -\sqrt{|SE_{E_o,E_m,Q_1} + SE_{E_o,E_m,Q_2}|} \\ \quad L_{E_o,E_m,Q}\text{の要素があり、和の値が負の場合} \end{cases}$$

演算子 $op$ が $\not\triangleright$、$\not\triangleleft$ の場合

$$SE_{E_o,E_m,Q_1 op Q_2} = \begin{cases} SE_{E_o,E_m,Q_1} - SE_{E_o,E_m,Q_2} \\ \quad L_{E_o,E_m,Q}\text{の要素が無い場合} \\ (SE_{E_o,E_m,Q_1} - SE_{E_o,E_m,Q_2})^2 \\ \quad L_{E_o,E_m,Q}\text{の要素があり、差の値が正の場合} \\ -\sqrt{|SE_{E_o,E_m,Q_1} - SE_{E_o,E_m,Q_2}|} \\ \quad L_{E_o,E_m,Q}\text{の要素があり、差の値が負の場合} \end{cases}$$

上の定義における $L_{E_o,E_m,Q}$ は以下のように定義する。

- 葉ノード (アルファベット $t$) の場合
  $L_{E_o,E_m,Q}$ = extent $E_m$ 中に含まれている $t$ の位置のリスト
- $Q_o$ の根ノードの場合
  $L_{E_o,E_m,Q_o} = \{E_o\}$ .
- それ以外のノードの場合
  $L_{E_o,E_m,Q_1 op Q_2} = L_{E_o,E_m,Q_1}\ op\ L_{E_o,E_m,Q_2}$

$Q_o$ の根ノードの場合に $L_{E_o,E_m,Q_o} = \{E_o\}$ とするのは、ある $Q_m$ とマッチする extent $E$ の中に複数の $Q_o$ とマッチする extent $E_1, E_2$ があった場合に、$(E_1, E)$ と $(E_2, E)$ のスコアに差をつけるためである。これにより、$(E_1, E)$ のスコアを計算するときには $Q_o$ とマッチする extent として $E_1$ のみを、$(E_2, E)$ のスコアを計算するときには $Q_o$ とマッチする extent として $E_2$ のみを使用する。

## 4 実験

3 節で説明したアルゴリズムを C++により実装し実験を行った。候補選択での $T$ の値を 1,000 とした。

以下の実験ではロボットで集められた Web ページ 13,325 ページ (185MB) を使用した。また実験は Pentium III 1GHz の計算機上で行われた。

実験結果はクエリの例およびその結果によって示す。

- クエリ:(“⟨title⟩” $\diamond$ “⟨/title⟩”) $\triangleright$ (“search” $\triangledown$ “retrieval”)
  - 検索時間 0.01 秒未満
  - 検索結果 (上位の出力)
    ⟨title⟩ acm programming contest file retrieval ⟨/title⟩
    ⟨title⟩ rick kazman's papers on computational linguistics/information retrieval ⟨/title⟩
- クエリ:((“⟨html⟩” $\diamond$ “⟨/html⟩”) $\triangleright$ “algorithm”) $\triangleright$ ((“⟨title⟩” $\diamond$ “⟨/title⟩”) $\triangleright$ (“information” $\triangle$ “retrieval”))
  - 検索時間 0.24 秒
  - 検索結果: 今回使用したデータにおいては (“⟨title⟩” $\diamond$ “⟨/title⟩”) $\triangleright$ (“information” $\triangle$ “retrieval”) にマッチする extent は存在しなかったが、“⟨html⟩” $\diamond$ “⟨/html⟩” の部分に “information” および “retrieval” の単語を含むものが出力された。

後者のクエリの実験でクエリと完全にマッチしていなくても出力されていることによりシステムの頑健性が示されている。本手法では候補選択を行うことにより検索対象が増えても検索時間はそれほど変わらず、本実験における検索時間は既存のシステムと比べても妥当であり、効率性が示されている。

## 5 まとめと今後の課題

本論文では Web 上のテキストに対するランク付き近接構造検索アルゴリズムを提案した。スコア計算において構造と単語の両方にスコアを与えることにより頑健なシステムを実現した。また、関連性が高い範囲のスコアのみを計算することにより効率的なシステムを実現した。

今後の課題としては、現在は最大範囲クエリを ⟨m⟩ $\diamond$ ⟨/m⟩ という形であると制限しているが、このような制限を行わなくても済むよう候補選択の方法を改善する必要がある。また、候補選択では演算子によらずクエリ中に含まれる単語の TF×IDF 値のみで候補が求められているが、クエリの演算子を満たす extent が優先されるよう改善したい。

## 参考文献


[1] Charles 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.

[2] Alistair Moffat and Justin Zobel. Fast ranking in limited space. In *ICDE*, pages 428–437, 1994.

[3] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.