文節重要度と係り受け整合度に基づく文要約アルゴリズム

小黒 玲　尾関 和彦　張 玉潔　高木 一幸
電気通信大学
{rei, ozeki, zhangbing, takagi}@ice.uec.ac.jp

1 はじめに

電子化テキストの爆発的増加に伴ってテキスト要約技術の必要性が高まり, この分野の研究が盛んになっている [1]. 従来の要約技術は, 重要度の高い文を抽出することにより, 文章全体の要約を行うものが多かったが, 目的によっては文単位の要約が必要な場合もある. そのため, 特にニュース字幕作成を目的として, 表層文字列の変換 [2] や, 変換規則の獲得 [3] などの研究が行われている. また, 重要度の低い文節や単語を削除することによって文を要約する手法も研究されており, 単語重要度と言語的な尤度の総和が最大となる部分単語列を動的計画法によって求める方法 [4] が提案されている. しかし, この方法では trigram に基づいた局所的な言語制約しか用いていないので, 得られた要約文が構造的に不自然となる可能性がある. 削除文節の選択に係り受け関係を考慮することで, 要約文の構造の保存を図る方法 [5] も研究されているが, この方法ではまず係り受け解析を行ない, 次に重要度の低い文節を削除するという, 二段階の処理が必要となる.

本稿では, 文の要約を「原文から, 文節重要度と文節間係り受け整合度の総和が最大になる部分文節列を選択する」問題として定式化し, それを解くための効率の良いアルゴリズムを提案する. この方法によって, 原文の部分的な係り受け構造が保存された, 自然で正確な要約文が効率的に生成できることが期待される.

2 問題の定式化

2.1 最適化問題

与えられた文節列から各文節の重要度と係り受け整合度の総和が最大になる部分文節列を選択する問題を考える. この問題は

- $w_0 w_1 ... w_{M-1}$: 与えられた文節列
- N: 選択する部分文節列の長さ ($N \leq M$)
- $p(m,n)$: 文節 w_mが文節 w_nに係るときの係り受け整合度 ($0 \leq m < n \leq M-1$)
- $q(m)$: 文節 w_mの重要度 ($0 \leq m \leq M-1$)

とし,

$$g(k_1, k_2, ..., k_{l+1}) \triangleq \begin{cases} q(k_1), l = 0 \\ \max_c \sum_{i=1}^{l} p(k_i, c(k_i)) + \sum_{i=1}^{l+1} q(k_i), 0 < l \end{cases}$$

とするとき, 関数 $g(k_1, k_2, ..., k_N)$ が最大になる文節番号 $0 \leq k_1 < k_2 < ... < k_N \leq M-1$ を求める問題として定式化できる. ここで c は部分文節列 $w_{k_1} w_{k_2} ... w_{k_{l+1}}$ 上の係り受け構造を表す. すなわち c は係り文節番号を受け文節番号に対応させる写像

$$c : \{k_1, k_2, ..., k_l\} \longrightarrow \{k_2, k_3, ..., k_{l+1}\}$$

であり, 以下の性質を持つ.

- 後方単一性: $k_m < c(k_m), (1 \leq m \leq l)$
- 非交差性: $m < n$ ならば $c(k_n) \leq c(k_m)$

この問題は文節集合列から最適文節列を選択するアルゴリズム [6] に準じた考え方により, 効率的に解くことができる.

2.2 再帰式

上で定義した関数 $g(k_1, k_2, ..., k_N)$ の最大化問題を解くために, その "部分問題" すなわち文節列長を $l+1 (l \leq N-1)$ とし, 先頭文節を "w_m", 最終文節を "w_n" に固定したときの最大化を考え, その時の最大値を

$$f(m,n,l) \triangleq \max_{\substack{m=k_1<k_2<... \\ <k_{l+1}=n}} g(k_1, k_2, ..., k_{l+1})$$

とおく. そうすると, $f(m,n,l)$ に関して図1のような再帰式が成り立つ. これは $g(k_1, k_2, ..., k_{l+1})$ の $m=k_1<k_2<...<k_{l+1}=n$ に関する最大化が, 変数列を $k_1, ..., k_{l'+1}$ と $k_{l'+2}, ..., k_{l+1}$ に分割することにより, $0 \leq l' \leq l-1, m \leq n' < m' \leq n, m=k_1<...<k_{l'+1}=n', m'=k_{l'+2}<...<k_{l+1}=n$ に関する最大化に帰着することから導かれる.

2.3 アルゴリズムの構成

図1の再帰式は $1 \leq l$ のとき, $l' < l$ となる $f(m,\cdot,l')$ と $f(\cdot,n,l-l'-1)$ が既に計算されていれば, 高々3つの

1. $m = n$ のとき: (lの動く範囲: $l = 0$ のみ)

$f(m,n,l) = q(m)$

2. $m < n$ のとき: (lの動く範囲: $0 < l \leq n - m$)

(a) $l = 1$ のとき

$f(m,n,l) = f(m,m,0) + f(n,n,0) + p(m,n)$

(b) $l = 2$ のとき

$f(m,n,l) =$

$$\max \begin{cases} f(m,m,0) + \max_{m<m'<n}\{f(m',n,1)\} + p(m,n); & (case1) \\ \max_{m<n'<n}\{f(m,n',1) + p(n',n)\} + f(n,n,0) & (case2) \end{cases}$$

(c) $3 \leq l \leq n - m$ のとき

$f(m,n,l) =$

$$\max \begin{cases} f(m,m,0) + \max_{m<m'\leq n-l+1}\{f(m',n,l-1)\} + p(m,n); & (case1) \\ \max_{0<l'<l-1,m+l'\leq n'<m'\leq n-l+l'+1}\{f(m,n',l') + f(m',n,l-l'-1) + p(n',n)\}; & (case2) \\ \max_{m+l-1\leq n'<n}\{f(m,n',l-1) + p(n',n)\} + f(n,n,0) & (case3) \end{cases}$$

図 1: 再帰式

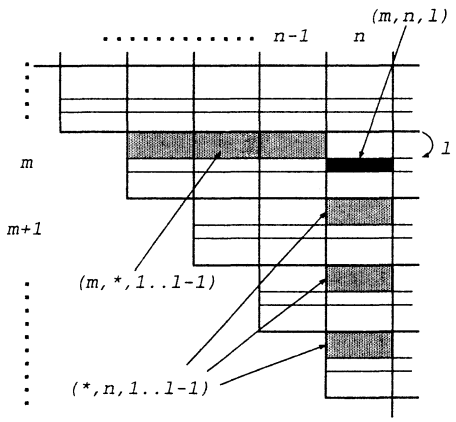

図 2: $f(m,n,l)$ を計算するとき参照される領域 (実際には制約条件も考慮する)

変数に関する最大化問題を解くことにより $f(m,n,l)$ が計算できることを表している. $f(\cdot,\cdot,0)$ は, 入力文節の重要度から直接計算できる. これから始めて, $l = 1$ のときには 2 文節の重要度とその間の係り受け整合度の和を計算し, $l = 2$ のときは $f(\cdot,\cdot,l)$ を 2 変数 m', n' が制約条件を満たす範囲で最大化を行ない, $3 \leq l$では 3 変数 m', n', l'が制約条件を満たす範囲で最大化を行なうという再帰的な処理によって, $f(m,n,l)$ を計算することができる.

以上の事実に注意すると, 図 2の様なテーブルを順次計算済の値で埋めていくことにより, $f(m,n,l)$ を計算するアルゴリズムが構成できる.

このとき, 係り文節 "w_m" は必ず受け文節 "w_n" より文頭側にあること, "w_m" と "w_n" 間を要約する場合, 要約後の部分文節列長が 2 つの文節間にある文節

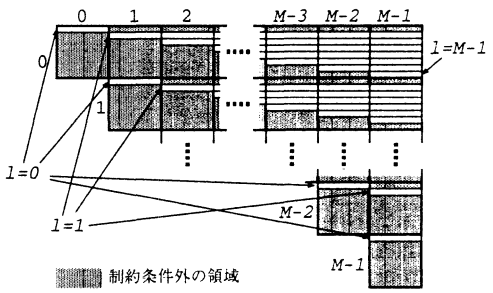

図 3: 制約条件を満たさない領域

数より大きくなることがないことから, 変数には以下のような制約が課せられる.

- $m \leq n$
- $m = n$ のとき $l = 0$
- $m < n$ のとき $0 < l \leq n - m$

これを図示したものが図 3である.

図 2のテーブルの升目を埋める順序に関する制約は, $f(m,n,l)$ を計算する際に $f(m,n',l')$ と $(m',n,l-l'-1)$ が制約条件の範囲で全て計算済であるということだけなので, その順序には大きな自由度がある. 変数 l,m,n を動かすとき図 4のように最外ループを n に関するループとすると, 入力に同期した処理の可能なアルゴリズムとなる. また, 最外ループを l に関するループとすれば, 文節数の少ない要約文から順に出力することが可能なアルゴリズムとなる.

2.4 バックトレース

最適部分文節列を求めるためには, 再帰式の右辺において最大値を与える変数の値を記憶する必要がある.

```
for n := 0 to M − 1
 begin
 for m := n downto 0 do
  begin
  if (m = n) then f(m, m, 0) := q(m);
  else
   begin
   for l := 1 to n − m do
    begin
    if (l = 1) then f(m, n, l) := ....
    else if(l = 2) then f(m, n, l) := ...;
    else f(m, n, l) := ....
    end;
   end;
  end;
 end;
```

図 4: 再帰式の実行 (入力文節同期)

これは最適文節分割点に対応するもので, 記憶する変数 $bp[m,n,l].lp, bp[m,n,l].np, bp[m,n,l].mp$ は, 最大値となる条件によって以下のようになる. すなわち, 最適文節分割点が図 1のどの場合であったかによって, 場合分けされる. ここで $bp[m,n,l].lp$ を lp と略すことにし, np, mp も同様とする.

1. $m = n$ の場合
   何も記憶する必要がない
2. $m < n$ の場合

(a) $l = 1$ の場合
$lp = 0, np = m, mp = n$

(b) $l = 2$ の場合
- *case*1
  $lp = 0, np = m, mp = m'$
- *case*2
  $lp = l - 1, np = n', mp = n$

(c) $l \leq 3$ の場合
- *case*1
  $lp = 0, np = m, mp = m'$
- *case*2
  $lp = l', np = n', mp = m'$
- *case*3
  $lp = l - 1, np = n', mp = n$

このバックポインタを用いて, 図 5で示した再帰関数 $out(m,n,l)$ によってバックトレースを行なうことにより, 最適部分文節列が得られる.

$f(m,n,l)$ の値は, 先頭文節と最終文節がそれぞれ文節 "w_m" と "w_n" であるとして, $l+1$ 文節で要約し

```
再帰関数 (⊕ は文字列の連接を表す)
 function out(m, n, l): char string;
  begin
  if(m = n) then out :="w_m",
  else
  begin
   l' := bp[m, n, l].lp;
   n' := bp[m, n, l].np;
   m' := bp[m, n, l].mp;
   out := out(m, n', l') ⊕ out(m', n, l − l' − 1);
  end;
 end.
```

図 5: バックトレース

たときの最大値である. したがって全文節列を N文節で要約するときには, まず $f(\cdot,\cdot,N-1)$ を最大とする m_o, n_oを探索する. この m_o, n_oを使ってバックトレースを開始する. すなわち最適部分文節列は, 以下のように求められる

$$(m_o, n_o) := \underset{m,n}{\operatorname{argmax}}\, f(m,n,N-1);$$

最適文節列 $:= out(m_o, n_o, N-1)$

また, 文献 [6] と同様にして, 文節列と同時に部分文節列の係り受け構造を出力することもできる.

2.5 計算量のオーダ

図 4のアルゴリズムを実行する際の加算回数 ($A(M)$) と比較演算回数 ($C(M)$) は次のようになる.

$$\begin{aligned} A(M) &= 1/360\times\textstyle\prod_{i=-3}^{2}(M+i) \\ &\quad +1/24\times\textstyle\prod_{i=-4}^{-1}(M+i) \\ &\quad +2/3\times\textstyle\prod_{i=-3}^{-1}(M+i) \\ &\quad +1/6\times(M-1)\times(M-2)\times(M+9) \\ &\quad +(M-1)\times M \end{aligned}$$

$$\begin{aligned} C(M) &= 1/720\times\textstyle\prod_{i=-3}^{2}(M+i) \\ &\quad +1/12\times\textstyle\prod_{i=-4}^{-1}(M+i) \\ &\quad +1/2\times(M-2)^2\times(M-1) \end{aligned}$$

したがって計算量のオーダは $O(M^6)$ である. これはかなり大きな計算量のように見えるが, 最高次の係数が小さいことと Mは高々40 程度までを考えておけばよいことから, 計算が困難になることはない. 実際, $M = 40$ の場合, 加算回数と比較演算回数は $A(40)\approx 1.1\times 10^7, C(40)\approx 5.4\times 10^6$ となり, アルゴリズムを C で実装し UltraSPARC-IIi(270MHz) 上で処理したときの計算時間は, 1 秒程度である.

表 1: 要約例

文節数:	文節要約文
原文:	年齢は まだ 十四だが 数えきれぬほど 日本の 舞台を 踏んだので 日本語は ぺらぺらだそうだ
8:	年齢は まだ 十四だが 日本の 舞台を 踏んだので 日本語は ぺらぺらだそうだ
7:	年齢は 十四だが 日本の 舞台を 踏んだので 日本語は ぺらぺらだそうだ
6:	十四だが 日本の 舞台を 踏んだので 日本語は ぺらぺらだそうだ
5:	日本の 舞台を 踏んだので 日本語は ぺらぺらだそうだ
4:	舞台を 踏んだので 日本語は ぺらぺらだそうだ
3:	踏んだので 日本語は ぺらぺらだそうだ
2:	日本語は ぺらぺらだそうだ
原文:	また 袖や 袖口 ポケット口などが 油汚れで 変色を おこす ことも あります
8:	袖や 袖口 ポケット口などが 油汚れで 変色を おこす ことも あります
7:	袖や 袖口 ポケット口などが 変色を おこす ことも あります
6:	袖や 袖口 ポケット口などが 変色を おこす ことも
5:	袖や 袖口 ポケット口などが 変色を おこす
4:	袖口 ポケット口などが 変色を おこす
3:	変色を おこす ことも
2:	変色を おこす

3 要約例

この手法を用いてATRコーパスの2文を処理した例を表1に示す. 本アルゴリズムで必要とされる係り受け整合度は, 文献[7]のペナルティに -1 を乗じたものとした. このペナルティ関数は学習コーパス中の係り受け距離の頻度分布を元に作成されている. また文節重要度は,

- 主部/述部や名詞/動詞を含む文節に形容詞や動作の程度や目的を表す文節より大きい値
- 文末の動詞には大きな値
- 形式名詞には小さな値

を設定した.

例にあげた2文は文献[7]で正しく係り受け解析できたものである. 2文どちらにおいても日本語として自然な出力が得られている. 特に1番目の例では係り受け整合度が有効に働き, 述語を中心とした文の構造が保たれている. 2番目の例では, 形式名詞 "こと も","あります" が削除され, 述語を中心とした文の構造は保たれていない. しかし, 重要度の低い形式名詞などを削除して簡潔に要約したいという目的には合っている. この場合は文節重要度が有効に働いたと考えられる.

4 おわりに

文節重要度と係り受け整合度に基づいて, 効率的に文の要約を行なうアルゴリズムを提案した. このアルゴリズムは要約文の係り受け構造も同時に出力できるので, 要約文を引き続き他言語に翻訳するときなどにも有用である. 計算量は原文文節数の6乗のオーダとなるが, 各変数の制約条件のために現実的な文節数に対して実行可能なものとなっている. また, 最外ループを制御する変数の選び方には自由度があるため, 即答性が求められる場合には文節の入力と同期して計算を進めるようにアルゴリズムを構成することが可能である.

今後は, 文節重要度や係り受け整合度の設定の仕方が要約結果に与える影響や, 文節重要度と係り受け整合度の適切なバランスなどについて検討し, 要約手法としての評価を行なう予定である.

参考文献

[1] 奥村学, 難波英嗣: "テキスト自動要約に関する研究動向 (巻頭言に代えて)," 自然言語処理 Vol.6, no.6, pp.1-26,1999.

[2] 若尾孝博, 江原暉将, 白井克彦: "テレビニュース番組の字幕に見られる要約手法," 情報処理学会研究会報告 97-NL-122-13, pp.83-89, 1997.

[3] 加藤直人, 浦谷則好: "局所的要約知識の自動獲得手法," 自然言語処理 Vol.6, no.7, pp.73-91, 1999.

[4] 堀智織, 古井貞熙: "話題語と言語モデルを用いた音声自動要約法の検討," 情報処理学会研究会報告 99-SLP-29-18, pp.103-108, 1999.

[5] 三上真, 増田繁, 中川聖一: "ニュース番組における字幕生成のための文内短縮による要約," 自然言語処理 Vol.6, no.6, pp.65-81, 1999.

[6] 尾関和彦: "係り受けの整合度に基づき最適文節列を選択する多段決定アルゴリズム," 電子情報通信学会論文誌 J70-D(3), pp.601-609,1987.

[7] 張玉潔, 尾関和彦: "文節間係り受け距離の統計的性質を用いた日本語文の係り受け解析," 自然言語処理 Vol.4, no.2 , pp.3-19, 1997.