大規模情報検索のためのダイナミックプログラミング手法

山本英子† 岸田正博‡ 武並佳則‡ 武田善行# 梅村恭司#

†通信総合研究所 ‡住友電工情報システム #豊橋技術科学大学

1. はじめに

ダイナミックプログラミング手法は情報検索に適用することができる。しかし、ダイナミックプログラミングによる文書間の類似度計測は文字の挿入や削除が行われても類似度を保持する能力を持つが、情報検索に利用する場合、客観的な精度が悪く、実行速度も遅いという問題がある。そこで、我々は精度を向上させるために、従来のダイナミックプログラミングを拡張した手法を提案した[7]。これは、文字ではなく、文字列を対象とし、一致した文字列がもつIDFを重みとして加算する手法である。しかし、この拡張によって情報検索精度は向上したが、実行速度が従来のダイナミックプログラミングと比べ、非常に遅い。そこで本研究では、重み付けを変更し、その結果から索引をあらかじめ作成することによって、ダイナミックプログラミング手法に分類される類似尺度でありながら、客観的な精度と実行効率を持つ尺度を提案する。

2. 検索精度の改善

ここでは、従来のダイナミックプログラミング手法から、情報検索精度を持つ手法への拡張方法を示す。

2.1. 単純編集類似度

編集距離はダイナミックプログラミングによって計算される代表的な尺度である[5]。これの距離はある文字列を他の文字列に修正する際に文字の編集が最小となるパターンを探し、そのときの編集回数を距離とする、相違を測る尺度である。そこで、この編集距離を反転し、類似尺度に変更した。この類似尺度は二つの文字列について、一致した文字の数を最大とするパターンを探し、そのときの文字数を類似度とする尺度である。この尺度を単純編集類似度(以下SIM1)と呼ぶ。次にSIM1を示す。ここで、α,βを文字列、x,yを異なる文字、""を空文字列とする。

- $SIM1("","")=0.0$
- $SIM1(x,y)=0.0$
- 先頭文字が一致するとき、$SIM1(x\alpha,x\beta)=MAX(SIM1(\alpha,x\beta),SIM1(x\alpha,\beta),SIM1(\alpha,\beta)+1.0)$
- 先頭文字が一致しないとき、$SIM1(x\alpha,y\beta)=MAX(SIM1(\alpha,y\beta),SIM1(x\alpha,\beta),SIM1(\alpha,\beta))$

2.2. 文字重み編集類似度

SIM1はどの文字についても一致すれば均等な重み1.0を加算する。しかし、検索において、ひらがなと漢字の貢献度は異なることが多い。これは、ひらがなは機能語を表すのに多く使われ、漢字は内容語を表すのに多く使われる。一般に有用なキーワードは内容語であるため、ひらがなと漢字の貢献度を同じとするのは不自然である。そこで、文字が一致した際、*1.0*ではなく、文字ごとに重み関数*Score*による重みを加算する尺度にSIM1を拡張した。この尺度を文字重み編集類似度(以下SIM2)と呼ぶ。重み関数*Score*については2.4節に示す。

2.3. 文字列重み編集類似度

SIM2は一致した文字ごとの重みを加算する。しかし、検索において、各文字の貢献度が小さくても、それらの文字の組合せによる文字列であることによって、大きく貢献することがある。そこで、一致する文字列ごとに重み関数*Score*による重みを加算する尺度にSIM2を拡張した。この尺度を文字列重み編集類似度(以下SIM3)と呼ぶ。次にSIM3を示す。ここで、$\alpha,\beta,\gamma,\delta,\xi$を文字列、$x,y$を異なる文字、""を空文字列とする。

- $SIM3("","")=0.0$
- $SIM3(\alpha,\beta)=MAX(SIM3_s(\alpha,\beta),SIM3_g(\alpha,\beta))$
- 先頭文字列が一致するとき、$SIM3_s(\xi\alpha,\xi\beta)=MAX(Score(\gamma)+SIM3(\delta\alpha,\delta\beta))$ ただし、$\xi=\gamma\delta$
- 先頭文字が一致しないとき、$SIM3_g(x\alpha,y\beta)=MAX(SIM3(\alpha,y\beta),SIM3(x\alpha,\beta),SIM3(\alpha,\beta))$

2.4. 重み関数

SIM2とSIM3では重み関数*Score*を使い、貢献度となる重みを計算する。次に*Score*を示す。ここで、ξを文字列、$df(\xi)$を文字列が出現する文書数、Nを文書数とする。

- 任意の文字列について、$Score(\xi)=-\log(df(\xi)/N)$

これはIDFと呼ばれる情報検索においてよく知られる情報量である。一般にIDFは単語のみを対象とする。しかし、本研究では単語だけでなく、任意の文字列を対象とする。SIM2では対象とする文字列の長さを1とし、文字重みとする。

3. 高速化

我々はこれまでに前節に示す三段階の拡張によって得たSIM3 を用いることで、ダイナミックプログラミング手法による情報検索の精度向上を図った。2.3 節に示した尺度SIM3を用いたダイナミックプログラミング手法は検索精度を持つが、時間的コストが非常に高い。そこで、本研究では、同じくSIM3 を用いるが、そこで使用する重み関数 Score を変更し、さらに、マッチングの対象となる文字列を制限するアルゴリズムを組み込むことによって、高速化を図る。本節では、変更した重み関数と、高速化のためのアルゴリズムを示す。

3.1. 重み関数の変更

高速化のために、すべての文字列に重み付けするのではなく、重み付けする文字列をすべて二文字でできた bigram と制限し、重みとしてIDF を与える関数に重み関数 Score を変更する。このことにより、重み付けにかかるコストを削減する。bigram に制限した理由は漢字を用いる言語における情報検索には文字の bigram でのマッチングが有効であることがよく知られており[4]、より長い文字列が一致する場合でも部分文字列である bigram の重みの合計でその文字列の貢献度を近似できると考えたためである。次に重み関数 *Score* を示す。

- 長さ 2 の文字列ならば、$Score(\xi) = -\log_2(df(\xi)/N)$

3.2. 高速化アルゴリズム

ダイナミックプログラミングにおいて、マッチングの対象となる文字列が少なければ、文書を走査するコストを削減できる。そこで、文書間の類似度への貢献が期待できる文字列を予め特定する。そうすることによって、検索時にそれらの文字列が出現する文書のみを考慮して処理することができる。次にアルゴリズムの概略を示す。

I. 文書集合の suffix array を作成する。
II. 各質問について、
  A) 長さ 2 の部分文字列(bigram)の集合を求める。
  B) その集合に含まれる bigram を文書集合に現れる頻度(Collection Frequency : cf)の低い順に並べ、上位から指定した数の bigram を取り出し、低い cf を持つ bigram の集合を作成する。
  C) その集合に含まれる bigram の文書集合上での出現場所について、
    i. 文書中と質問中での位置を登録しながら、
    ii. その bigram を含む文書の集合を求める。
III. 質問毎にⅡ.C. ii で作成される文書集合について、
  A) 登録された位置情報を利用して、SIM3 で質問と各文書の類似度を計算する。
  B) 類似度が高い上位 1000 文書を検索結果とする。

提案するダイナミックプログラミング手法は行程Ⅱ.B で集めた低い cf を持つ bigram を索引語として利用する。このことにより、検索に役立つ用語を索引語として利用する通常の情報検索手法と同様、類似度計算の対象となる文書を予め制限することによって、実行速度を改善する。

従来の検索システムは索引語が出現する文書とその文書内頻度を登録した表を利用し、類似度計算を行う。一方、我々が提案する手法には索引語の頻度情報の他に、ダイナミックプログラミングに従い類似度を計算するため、類似度に貢献する索引語の出現場所の情報が必要である。本研究では、出現場所の情報を得るために、Suffix Array[1]というデータ構造で文書集合を参照する。このデータ構造はデータに含まれる文字列の場所を特定できる構造を持つ。図 1 に Suffix Array の例を示す。Suffix Array は、文書数を N とすると、$O(Nlog(N))$で作成し、任意の文字列の出現場所を $O(log(N))$で特定できる。

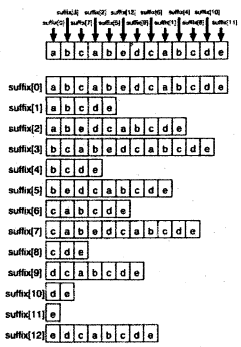

図 1 Suffix array と文字列の出現場所

4. 実験

本節では、2 節に示した SIM1、SIM2、SIM3、3 節に示した重み関数とアルゴリズムを用いた提案手法(以下 FDP)を、検索精度、記憶領域、実行速度の三つについて比較する。

実験には、学術論文抄録コーパスと検索課題 83 問(このうち 30 問は訓練用)、正解データを収めた NTCIR1[3]を用いた。検索課題にはユーザの質問を短く表した「検索要求」とより詳細に表された「検索要求説明」が含まれている。一般に、検索のためにシステムに入力される質問はキーワードの列挙か短い文である。このことから、実験では、質問に「検索要求」の部分を用いた。計算機環境は AMD Athlon MP 1900+ の dual CPU、3GB Memory、OS は TurboLinux7 である。

4.1. 検索精度

提案手法である FDP では、マッチングに貢献できる文字列を制限する。貢献できる文字列を制限することは検索精度の低下を招きかねない。そこで、本節では FDP が SIM3 の検索

精度を保持できるかどうかを検証する。また、制限数の違いによって、保持できるかどうかも異なる。そこで、制限数を5, 10, 15, 20, 30, 50としたFDPを検証の対象とする。

表1, 2にそれぞれ訓練用(topic01-30)と本番用(topic31-83)の課題について検索精度による性能比較を示す。表中に現れるFDPに続く数字はそれぞれ貢献できる文字列の数である。表1, 2ともに、平均精度、R-精度ともにFDP20が最も高く、また、FDP5,FD10を除き、他のFDPはSIM3より高い精度を示した。このことから、FDPは類似度に貢献できる文字列を数十個に制限するにも関わらず、検索精度を保持していることがわかる。

表 1 精度による性能比較(topic01-30.desc)

システム	平均精度	R-精度
SIM1	0.1349	0.1790
SIM2	0.1948	0.2296
SIM3	0.2691	0.3024
FDP5	0.2547	0.2649
FDP10	0.2948	0.3089
FDP15	0.3109	0.3446
FDP20	0.3207	0.3574
FDP30	0.3176	0.3421
FDP50	0.3131	0.3377

表 2 精度による性能比較(topic31-83.desc)

システム	平均精度	R-精度
SIM1	0.0545	0.0845
SIM2	0.1245	0.1596
SIM3	0.1807	0.2083
FDP5	0.1277	0.1505
FDP10	0.1766	0.2013
FDP15	0.2144	0.2280
FDP20	0.2398	0.2621
FDP30	0.2353	0.2485
FDP50	0.2354	0.2488

4.2. 記憶領域

ダイナミックプログラミング手法では類似度が最大になるマッチングパターンを見つけるために、各パターンで考慮した文字列の場所を記憶する必要がある。このため、記憶領域へのアクセスは最多だと文書集合の文字数と質問の文字数の積、必要と考えられる。しかし、FDPを用いると、類似度に貢献できる文字列を予め制限するので、この記憶領域はそれほど大きくならない。記憶領域へのアクセス回数は、言い換えると、類似度に貢献できる文字列の出現頻度(cf)の総数である。

図2にSIM3で考慮されるすべての部分文字列(AllNgram)と、FDPで数を制限しなかった場合に考慮されるすべてのbigram (AllBigram)、FDP20で選ばれた20個のbigram (20Bigram)について、質問毎の出現頻度総数を示す。

図2より、どの質問についても、AllBigramと20BigramはAllNgramに比べ、cfの総数が桁違いに低いことがわかる。これは、FDP20はSIM3に比べ、メモリアクセスが非常に少ない、つまり、必要となる記憶領域が小さいことを意味する。このことにより、FDPを用いることによって、メモリが小さい計算機においても大規模な情報検索が可能であると考える。

また、図3にAllBigramと20Bigramに関するcfの総数を示す。図3より、制限数20という設定は、多くの質問についてすべてのbigramを考慮しているが、完全ではないことがわかる。一般に、処理を簡略化すると、性能が低下する。しかし、FDPは4.1節に示すように、FDP20はもっとも高い精度を保持し、考慮する文字列が多ければ精度が高いわけではない。このことから、FDPはcfの低い文字列を選ぶことによって、貢献度の高い文字列を選択できていると考える。

4.3. 実行速度

4.1節と同じ条件で、検索対象とした文書数の増加による実行時間の変化を観察した。ただし、SIM系統はC言語、FDP系統はJava(JDK1.3.1.04)で実現したシステムを用いて計測した。前処理のSuffix Array作成には、FDPはSIMより1.5倍時間がかかる。このように、同じアルゴリズムであれば、JavaはC言語より実行速度が一般に遅い。

図4, 5にtopic01-30とtopic31-83について検索にかかった時間を示す。横軸は検索対象とした文書数、縦軸は実行時間とする。しかし、図4, 5に示すように、すべてのSIMはどのFDPと比べても、実行時間が非常にかかる。このことから、3節に示すアルゴリズムは大幅に実行速度を改善すると証明される。また、検索対象となる文書数が増加しても実行時間が指数的に増加しないことがわかる。これは、FDPが大規模情報検索に有効なダイナミックプログラミング手法であることを示唆している。

5. 関連研究

本研究の提案手法はダイナミックプログラミング手法の一つである。ダイナミックプログラミング手法の適用分野として、もっとも代表的な分野は遺伝子情報に関する研究である。これは、ダイナミックプログラミングが遺伝子情報のマッチングに有効であることが知られているためである。そこで、本研究では、文書を対象とする情報検索にダイナミックプログラミングを適用した。その結果、情報検索精度を持つシステムを作成できたが、処理速度が遅かった。近年、遺伝子情報検索においても、高速なシステムが求められている。そこで高速な遺伝子情報検索システムBLAST[6]が開発された。BLASTは特徴的な遺伝子配列を特定するヒューリスティックを用いることで高速化を実現しており、ダイナミックプロ

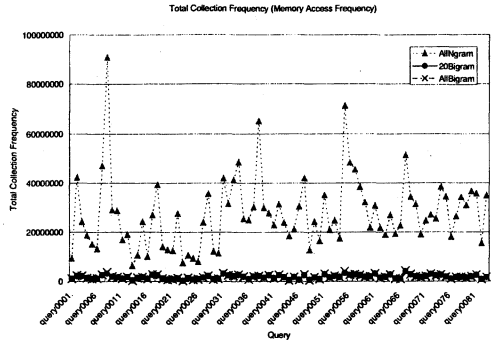

図 2 メモリアクセス回数(出現頻度総数) の比較

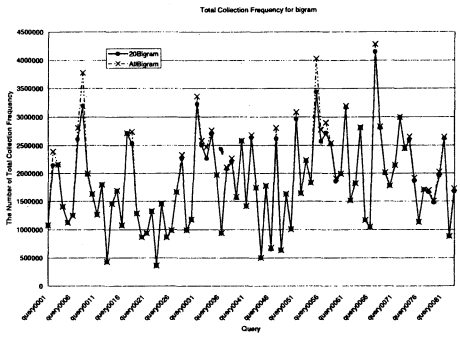

図 3 制限数の違いによる出現頻度総数の比較

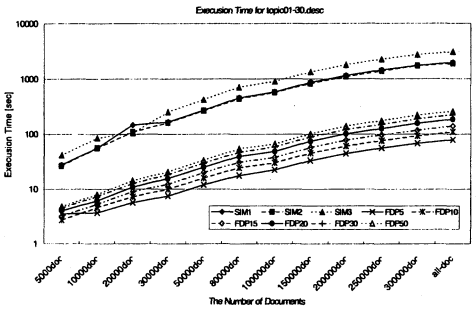

図 4 実行時間による性能比較(topic01-30)

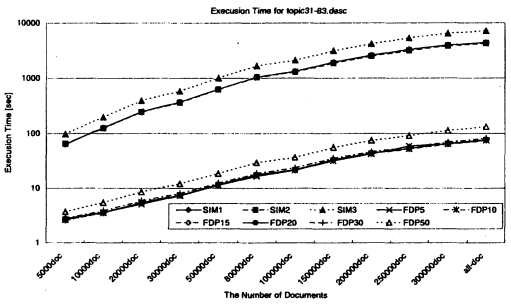

図 5 実行時間による性能比較(topic31-83)

グラミングではない。一方、本研究では、高速化をダイナミックプログラミング手法に基づき実現した。

また、音楽を対象とした情報検索において、採譜の誤りがデータの欠損や挿入につながる。このため、この研究分野においてもダイナミックプログラミング手法に基づく検索システムが構築されている[2]。音楽を表現する記号は文書表現に使われる文字より少ないため、検索対象は長く大きいものと考えられる。このため、文書を対象とする情報検索よりも手法のスケーラビリティが問題となる。本研究で提案するダイナミックプログラミング手法は、図4, 5に示すように文書数の増加しても、通常の手法に比べ、実行時間の増加は緩やかであることがわかる。このことから、提案手法はスケーラビリティを備えていると考える。

6. おわりに

本研究では、高速なかつ大規模情報検索のためのダイナミックプログラミング手法を提案した。この手法は、高速化を実現するために、検索に貢献する文字列を選別する。これによって記憶領域の大きさとアクセス回数を削減できる。これらの作用によって、精度は変わらず、実行速度が大きく向上し、検索対象が大規模なものであっても対応できるスケーラビリティを備え得ると報告した。

参考文献

[1] Manber U. and Myers G., "Suffix arrays: a new method for on-line string searches.", *SIAM Journal of Computing*, Vol.22, No.5, pp.935—948, 1993.

[2] Ning Hu and Roger B.D., "Comparison of Melodic Database Retrieval Techniques Using Sung Queries.", In Proceedings of JCDL 2002, pp.301—307, 2002.

[3] NTCIR Project: http://research.nii.ac.jp/ntcir/.

[4] Ogawa, Y. and Matsuda, T., Overlapping statistical word indexing: A new indexing method for Japanese text, In *Proceedings of SIGIR97*, pp.226-234, 1997.

[5] Robert R. Korfhage, "Information Storage and Retrieval.", In *WILEY COMPUTER PUBLISHING*, pp.291—303, John Wiley & Sons, Printed in USA, 1997.

[6] Setubal J.C. and Meidanis J., 五条堀孝 監訳, "分子生物学のためのバイオインフォマティクス入門", 共立出版, 2001.

[7] 山本英子 武田善行 梅村恭司, "情報検索のための表記の揺れに寛容な類似尺度", 自然言語処理, Vol.10, No.1, pp-63—80, 2003.