文字データの特徴を捉えた類似文字列検索に関する研究

勝俣, 彰文 / KATSUMATA, Akifumi

(発行年 / Year)
2011-03-24
(学位授与年月日 / Date of Granted)
2011-03-24
(学位名 / Degree Name)
修士(工学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

2011年度　修士論文

文字データの特徴を捉えた類似文字列検索に関する研究

STUDIES ON APPROXIMATE STRING MATCHING
CAPTURING FEATURES OF STRING DATA

指導教官　三浦 孝夫 教授

法政大学大学院 工学研究科
電気工学専攻 修士課程

09R3105　勝俣 彰文
Akifumi KATSUMATA

目 次

第1章　序論

1.1　問題の背景

近年のインターネット技術の発展などにより，我々が得られる情報量は膨大になり，探したい情報のほとんどがWeb 上に存在する時代となっている．しかしその驚異的な増加に伴って，単純に真に獲得したい情報に辿り着くには困難になり，計算機による情報検索技術の高度化が求められている．

テキスト文字列から目的の文字列を探索する文字列検索問題は，情報検索における最も代表的な問題の一つである．文字列検索は質問文字列に一致した文字列を探索する“ 完全一致文字列検索 ”と，一部の異なりを許す“ 類似文字列検索 ”の 2 種類に分類される．インターネット上には膨大な量の Web ページが存在するが，異なる Web ページの文書では同じ意味の言葉が表記揺れを起こしていることが多い．表記揺れの言葉は，完全一致文字列検索では検索できないため，類似文字列検索が有用である．また，コンピュータウィルスの検出に対しても文字列検索技術が用いられているが，あるウィルスが公開された後に，その一部を改編した亜種が出現することが多い．亜種のデータについて詳細にわからない状況でも事前に検出することが望まれるため，完全一致文字列検索ではなく類似文字列検索の技術を利用する．この他にも，DNA 配列データに対して特定文字列検出等を扱うバイオインフォマティクス分野や，音声文字列と自然言語で扱う文字列の対応関係を調べる音声認識分野など，類似文字列検索の手法は幅広く応用されている．

近年ではデータベースにおいて XML データによる管理が行われていることも多い．その検索においても類似文字列検索を適用することができる．しかし文字列は基本的に 1 次元データであるが，XML 構造などの階層情報に関しては，多層的な並びを探索することから単純に 1 次元データとしての探索処理を適用できない．立体的な文字データとして捉えた上での文字列類似検索を考える必要がある．

また，文字列検索の最も一般的な応用例として，検索エンジンによるWebサイト検索がある．Google では主に質問文字列がWebページ上の文書における重要キーワードと一致するものを検索結果として出力する．しかし，利用者の入力した質問文字列にエラーがある場合，検索結果が得られないことがある．その場合，類似したキーワードの中で頻度の高いものを真に意図したキーワードとして推定し，再検索を促す．この推定に対しては，予め検索エンジン側で利用される各キーワードについての利用頻度を記憶しておく必要があり，容易にこの手法を用いるためには領域を節約する頻度の記憶手法が求められる．

1.2 扱う問題

本研究では，まず XML データ等の階層情報を扱う文字データ，つまり多次元空間上に発展させた文字データに対する類似文字列検索手法を提案する．しかし，多次元文字データを質問データとする類似検索は困難なことに加え，ユーザーが質問データと検索解との類似性や意味を捉えることは容易ではない．その点を考慮して，本研究では空間データに対する文字列の類似検索を実現する．

上記で提案する多次元での類似検索問題では，文字列の類似度計算に対して従来提案されている文字列距離を用い，類似検索の重要性を示す．従って，本研究では検索性能の向上についても考え，文字の遷移確率を考慮した文字列距離による類似検索手法を提案する．この手法は多次元環境だけでなく一般的に扱われる 1 次元文字データに対しても有用であると考えられるため，本稿では評価しやすい 1 次元文字データに対して実験を行い，有用性を評価する．

1.2.1 多次元文字データに対する類似検索

2 次元データでは,「任意の 2 点間の距離、方向」などの空間構造を意識した情報の広がりを表せるという意味で，情報のモデル化能力が向上する．多次元データはさらに一般化され，対象となる情報の記述力は増加する．一方，1 次元データで単純に扱われていたエディタ操作では線形探索や正規表現探索，あるいは部分文字列に対する挿入・削除・置換という操作機能しか提案されていない．多次元空間で要求される操作とはどのようなものなのか，この実行に要する負荷はどれくらいなのかなど，具体的な考察が必要である．

本研究では，3 次元空間上で立体的に文字データを扱うことが可能な 3 次元テキストエディタを用い，それにより 3 次元文字データを編集・構築する．そのデータから断面となる 2 次元文字データを選択し，列・行方向に存在する文字列に対して類似検索を行うことで，多次元データからの類似文字列解の抽出を実現する．

1.2.2 文字遷移確率を用いた検索による類似解の自動推定

質問文字列に類似する文字列を検索する上で，質問文字列と検索対象となる文字列間の類似度の計算が必要である．類似度は文字列距離によって計算でき，よく用いられるのが編集距離やハミング距離である．これらの距離関数で距離を計算した場合，同じ距離の文字列が多数出現することがあり，一意な検索解を得るには，文字列距離によって類似度を測ることに加えて文脈情報等の付加情報により類似度をより明確な差を得ることが必要不可欠である．

本研究では，文字列距離のみで類似度を詳細に計算する手法を提案する．文字の異なる数を単純に文字列距離とした従来手法に対し，本手法では文字の異なりを文字列

内の文字遷移確率の変化と捉えて文字列距離に適用する．提案した文字列距離により，容易に類似検索解を獲得することを実現する．

また，バイオインフォマティクス分野では DNA 配列等の長大文字列に対する類似検索手法が用いられていて，BLAST などの高速検索アルゴリズムが存在する．本研究では，このアルゴリズムを英単語等の比較的短い文字列に適応させた手法に関しても論じ，その有用性の評価を行う．

1.2.3 時系列文書データに適応した類似検索

上記で示した類似検索手法では，q-gram の頻度をコーパス内の全文書から得て，文字遷移確率に利用する．しかし，ニュース記事等の文書は時刻によって内容が変化するため，同じ質問文字列でも検索ユーザーの意図する類似検索解は時刻ごとに異なるはずである．検索ユーザーの最も意図するためにはコーパスの文書から確率を得るだけではなく，時系列文書から得た q-gram 頻度による文字遷移確率も利用して文字列距離の動的更新を行うことが必要不可欠である．

本研究では，上記で提案した遷移確率距離に対し，動的更新を行う手法を提案する．時系列文書に対応させることで遷移確率距離をより有用なその提案手法による類似検索が従来手法・前提案手法に対してどれだけ有用であるか比較評価を行う．

1.3 論文の構成

本研究では，以上の問題について以下の構成で論じる．第 2 章では，多次元に発展させた文字データに対する文字列近似照合手法について論じる．第 3 章では，確率的な文字列距離を用いた類似文字列検索手法について論じる．第 4 章では，更新型確率距離による文字列の類似検索手法について述べる．第 5 章で結論とする．

1.4 発表論文

1. 勝俣彰文, 三浦孝夫: “2 次元空間上の文字データの近似照合”，電子情報通信学会 2009 年総合大会 学生ポスターセッション, 2009.
   多次元空間上から断面抽出した 2 次元文字集合に対する近似照合を行う．2 次元文字データで列・行方向に同時にエラー数を確認する DP アルゴリズムを提案し，その有用性を検証する．

2. 勝俣彰文, 三浦孝夫: “Spatial Approximate String Matching”, *Pacific Rim Conference on Communications, Computers and Signal Processing* (*PACRIM*), pp.123-128, 2009.

多次元空間上から断面抽出した 2 次元文字集合に対する近似照合を行う．2 次元文字データで列・行方向に同時にエラー数を確認する DP アルゴリズムを提案し，その有用性を検証する．

3. 勝俣彰文，三浦孝夫: “遷移確率距離を用いた類似文字列検索”, 電子情報通信学会 知能ソフトウェア工学研究会 (*KBSE*) 110(61), pp.27-32, 2010.
   文字列中の文字の遷移確率を用いた類似検索により，ユーザーの意図に最も合致する解の自動推定を行う手法を提案する．また，ホモロジー検索アルゴリズムに対応する高速検索手法も提案し，有用性を評価する．

4. 勝俣彰文, 三浦孝夫，塩谷勇: “Approximate String Matching Using Markovian Distance”, *Third International Symposium on Parallel Architectures, Algorithms and Programming* (*PAAP*)，pp.231-238，2010.
   文字列中の文字の遷移確率を用いた類似検索により，ユーザーの意図に最も合致する解の自動推定を行う手法を提案する．また，ホモロジー検索アルゴリズムに対応する高速検索手法も提案し，有用性を評価する．

5. 勝俣彰文, 三浦孝夫: “動的確率距離による類似文字列検索”, 電子情報通信学会 2011 年総合大会 学生ポスターセッション，2011.
   遷移確率距離を時刻に応じて動的に更新する手法を提案する．この手法により，時系列文書により時々刻々と変化する現在情報にも対応した類似検索を実現する．

第2章　2 次元空間上の文字データの近似照合

本研究では 2 次元上に展開された文字の集合に対し，最大 k 回までの違いを許す近似照合方法の提案と実験による性能評価を行う．これまで提案されたアルゴリズムでは，1 次元データ（文字列）に対して k 個以内の違いを許容しながら最長部分文字列を照合する．本稿では，2 次元文字データで列・行方向に同時にエラー数を確認する DP アルゴリズムを提案し，その有用性を検証する．

2.1　前書き

テキスト文字列（1 次元データ）からパターン文字列に類似した文字列（k 個の違いを許す照合）を探索することを近似文字列照合という．2 文字列間の違い（距離）は編集距離により決定し，許容エラー数 k 以内であれば類似とみなす．例えば *Bath* と *Birth* では a と ir の部分が異なり，距離 2 となる．しかし，近年研究されている XML 構造などの階層情報では，多層的な並びを探索することからこのような 1 次元データ照合処理を単純に適用するわけにはいかない．さらに，遺伝子情報処理では，ゲノムの塩基配列を決定するため，断片的な DNA 塩基配列を完全な DNA 配列に構成する作業が行われる．このとき，複数の断片で重複して決定される部分を検出し，さらに DNA 配列の繰返し構造を考慮しつつ，つなぎ合わせる操作が必要となる．DNA 断片を分離するには 1 次元データの近似照合では不十分であり，空間的な関連性を扱う近似照合が必要である．

本研究ではこの照合法を文字面（2 次元データ）に拡張し，空間的な文字列類似探索（2 次元近似文字列照合）を論じる．このため，本研究ではまず **3 次元テキストエディタ** を開発し，編集した文字データを 3 次元空間 に格納する環境を構築する．その格納空間から，利用者がある ”断面” を選択することでその断面の文字データ（文字面）をエディタウィンドウ上に表示することが可能である．この表示する文字面に対して列・行方向に並ぶ文字を文字列として扱い，パターンとの近似照合を行うことにより今回の照合法を実現する．

テキストファイルなどの従来の文字データに対し 2 次元近似文字列照合を行うには一度 2 次元配列上に文字データを格納してから照合という手順を踏まなければならない．

しかし 3 次元テキストエディタを用いることで文字データは編集と同時に 3 次元空

間に格納することが可能である．つまり，照合の度に多次元配列上に格納する手間を省くことができる．

本論文の構成は次の通りである．第 2 章では ”3 次元テキストエディタ” について述べ，開発したシステムの構造を示す．第 3 章で今回の照合法に適用した動的計画法（DP）による近似文字列照合について述べ，第 4 章で本研究で提案した 2 次元近似文字列照合法について述べる．その後第 5 章で実験を行い，得られた結果による有用性を示し，最後に第 6 章で結論とする．

2.2 3 次元テキストエディタ

本章では今回開発したシステムである 3 次元テキストエディタについて述べる．

従来のテキストエディタではエディタウィンドウ上で文字データを扱いながら編集を行う．しかし利用者に見えるエディタウィンドウとは 2 次元であるため，3 次元的に文字データを扱うには文字データを格納する空間を別に用意する必要がある．従って 3 次元テキストエディタでは，文字データ格納のための 3 次元空間 ”3 次元テキストエリア” と，文字データの編集を行うエディタウィンドウである ”平面ウィンドウ” を併せ持つ．利用者は ”平面ウィンドウ” を通して ”3 次元テキストエリア” 内にある文字データを編集することで，3 次元文字データを編集することが可能である．以下では 3 次元テキストエディタを構成する ”平面ウィンドウ”・”3 次元テキストエリア” について概説する．

2.2.1 3 次元テキストエリア

3 次元テキストエリアは，Java における可変配列機能を用いて作成した 3 次元の文字データ格納領域である．配列は本来その配列サイズを決めてからその中に要素を入れていくため，サイズが限定される．しかし，可変配列は配列サイズを増減することが可能な配列であり，後から要素を挿入した場合でも内部で配列サイズを増加する．これを用いることで，データ保管領域のサイズの拡張が可能であり，平面ウィンドウによる編集で文字データが次々にこの領域に格納された場合にも対処することができる．

この可変配列は 1 次元の配列であるため，これにより生成する ”LineList”，”PageList”，”BookList” という 3 種類のオブジェクトを入れ子にすることにより，本の形をイメージする 3 次元のデータ保管領域を作成している．”BookList” という本に対し，”PageList” というページが 1 ページ，2 ページ，…と複数存在し，さらに各 ”PageList” に対し ”LineList” が 1 行，2 行，…と複数存在するといった具合である．3 次元テキストエリアの領域拡張は，PageList・LineList を BookList の中に追加することにより行う．

3 次元テキストエリアへの文字の格納は，平面ウィンドウにおいて文字が入力されたときに行われる．平面ウィンドウによる文字入力が起きたとき，文字の位置情報（列，

行，ページ）も 3 次元テキストエリア側に渡される．位置情報は 3 次元テキストエリアでの 3 次元座標（X, Y, Z）に変換し，その座標に則した位置に文字を格納する．3 次元テキストエリアにおいての 3 次元軸 X，Y，Z はそれぞれ LineList，PageList，BookList が展開する方向を示す．例えば図 2 のように ”c” という文字が（3, 2, 4）という座標を持ち, 3 次元テキストエリア内に挿入されたとき，座標（3,2,4）に挿入するということは,

「BookList 内にある 4 番目の PageList を取り出し，さらにその中の 2 番目にある LineList を取り出す．文字”c”はその LineList における 3 番目に挿入」
ということを表している．

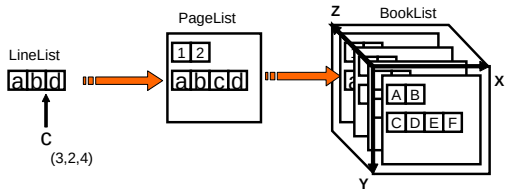

図 2.1: 格納領域 ”3 次元テキストエリア” の内部構成

2.2.2 平面ウィンドウ

平面ウィンドウとは従来のテキストエディタにおける文字データ編集画面である．異なる点は 3 次元テキストエリアから文字データを平面的に[1] 取り出し文字面編集を行うという点である．3 次元軸 X/Y/Z があり，ウィンドウの横軸に 1 つ，縦軸に異なる 1 つ選ぶ．それによりできる平面は X-Y , Y-X , Y-Z , Z-Y , Z-X , X-Z の 6 通りとなる．平面ウィンドウはその 6 通り分存在し，各平面に対応する 3 次元テキストエリアの断面文字データを切り出して・表示する．X-Y，Y-X は 3 次元テキストエリアの PageList に沿ってスライスした断面を表示し，Y-Z，Z-Y は横にスライスした断面，Z-X，X-Z は PageList に垂直になるように縦にスライスした断面となる．3 次元テキストエリアにおいて１つ目の PageList に対し ab/cdef，2 つ目の PageList に AB/CDEF という文字データが入って格納されていたとする (/は改行を表す)．このとき X-Y，Y-Z，Z-X ウィンドウを開いたときの表示文字データは図 3 のようになる．

また平面ウィンドウには数値を増減できるスピナーボタンを設置している．内部の数値を増減することにより，平面ウィンドウにおける ”ページ” を変更する．ページ数

[1]平面的に，と書いたのは実際に平面ウィンドウ上に表示している文字データは 1 次元であるからである．従来のテキストエディタでは文字列のような 1 次元文字データの中の「改行文字」により，文字列を行ごとに分割表示する．平面ウィンドウにおいても 3 次元テキストエリアから指定した平面から文字データを取り出すが，実際には平面ウィンドウで指定された横軸方向にまず文字列を取り込み, 改行を加え，その次の行に移って再び読み込み，改行を加え，…の繰り返しによってできた文字列データを平面ウィンドウで表示している

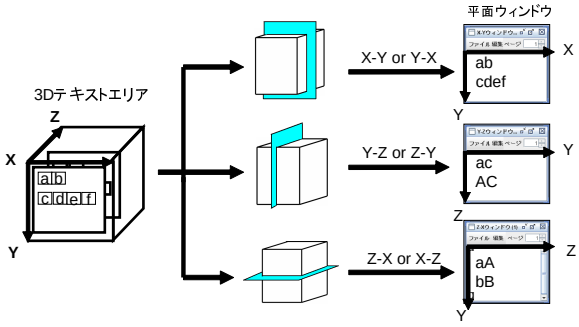

図 2.2: 各平面ウィンドウでの表示例

により，表示する断面データを 3 次元テキストエリア上で前後に移動することが可能である．例えば X-Y ウィンドウであれば，これにより Z = 1, 2, 3,⋯ のときの X-Y 平面文字データを取り出して表示できる．

2.3 動的計画法による近似文字列照合

2 文字列が近似するかを調べるには，その 2 文字列間での距離（エラー数）を調べる必要がある．距離は編集距離という距離関数に基づき決定する．本章では文字列間における編集距離（Edit Distance）について述べ，その編集距離を求めるための動的計画法（Dynamic Programming , DP）について概説する．

2.3.1 編集距離

2 文字列 $X_{1...n} = x_1x_2...x_n$ と $Y_{1...m} = y_1y_2...y_m$ が与えられたとき，この 2 文字列の距離を $ed(X, Y)$ と表す．距離は x を y に一致させるための最小操作コストを表している．操作には置換・削除・挿入があり，操作を 1 回行うごとにコストが 1 ，つまり距離が 1 増加する．例えば $X = zcde$ ，$Y = abcd$ としたとき，X と Y の距離 $ed(X, Y)$ を求めると，

$zcde \rightarrow acde$ (置換) $\rightarrow abcde$ (挿入) $\rightarrow abcd$ (削除)

というような操作手順により X を Y の形に一致させることが可能であり，$ed(X, Y)$ は 3 となる．

2.3.2 動的計画法を用いた編集距離の計算

動的計画法により，2 つの文字列 $X_{1...n}$ と $Y_{1...m}$ における距離 $ed(X, Y)$ を編集距離に基づき計算する．行に X ,列に Y を配置した行列 $C_{0...n,0...m}$ を作成し，その行列に以下の 3 つの式に基づいて値を埋めることにより $ed(X, Y)$ を求めることができる．

$$C_{i,0} = i \tag{2.1}$$
$$C_{0,j} = j \tag{2.2}$$

$$C_{i,j} = min \begin{cases} C_{i-1,j-1} + \delta(x_i, y_j) \\ C_{i-1,j} + 1 \\ C_{i,j-1} + 1 \end{cases} \tag{2.3}$$
$$※ if(t_i = p_j) \quad \delta(t_i, p_j) = 0$$
$$else \quad \delta(t_i, p_j) = 1$$

(1)，(2) 式は「長さ 0 の文字列と長さ $i(j)$ の文字列の距離は $i(j)$ である」ということを表す.
(3) 式は行列における左上・左・上の値があらかじめ計算されている時，それらの値に操作コストを加えることによって $C_{i,j}$ が求まることを表す．$C_{i-1,j-1}$ から遷移した場合，比較文字列に文字 x_iと y_j が追加される．x_iと y_j が一致していれば行列値 $C_{i,j}$ は $C_{i-1,j-1}$ と等しく，そうでなければ $C_{i-1,j-1}$ に置換コストとして 1 を追加したものが $C_{i,j}$ に入力される．$C_{i-1,j}$ から遷移した場合は x_i が削除されていることを表し，$C_{i-1,j}$ に削除コスト 1 が追加される．$C_{i,j-1}$ から遷移した場合は y_j が挿入されていること表し，$C_{i,j-1}$ に挿入コスト 1 が追加される．これらの中で，コスト和が最小となる値が最終的に $C_{i,j}$ となる.

計算によって得られた行列値 $C_{i,j}$ は $X_{1...i}$ と $Y_{1...j}$ の距離を表している．したがって行・列が共に末尾である行列値 $C_{n,m}$ が 求める距離 $ed(X, Y)$ になる．この値が許容エラー数 k 以下の値であればこの文字列同士は近似していることになる.

2.3.3 長いテキストに対する編集距離の計算

動的計画法による編集距離の計算はパターンに類似した文字列を長いテキスト中から部分的に探す近似照合法に対しても適用可能である $X = T, Y = P$（T :テキスト，P :パターン）として同様に計算を行う．しかし今回は先ほどの (1) 式を $\underline{C_{i,0} = 0}$ と変更する．この変更により，テキストの任意の位置から照合を開始することが可能になる.

P に対して，T 内のどの部分文字列が近似するのかは最下行の行列値により判定する．最下行の行列値から許容エラー数 k 以下の所を探索し，k 以下であった場所から単調減少していくパスを左上・左・上方向へ辿る．値が同じであり，進みたい方向が複数になる場合には，優先的に左へ辿る．そのパスの間に位置するテキスト部分文字列がパターンに近似した文字列である．図 1 は長いテキストに対してパターン "ABC" を与え，許容エラー数 $k = 1$ 以内で近似する部分がないかどうかを動的計画法により探索している．計算の結果，最下行に $k = 1$ 以下の部分が 2 つ出現している．その 2 点からパスを辿ることにより近似文字列 AIBC，AC を見つけ出している.

図 2.3: 動的計画法による編集距離計算 (テキストが長い場合)

2.4 2 次元文字列近似照合

本章では本研究の目的である 2 次元近似文字列照合について述べる．2 次元近似文字列照合は第 3 章で述べた 3 次元テキストエディタに，第 2 章で述べた動的計画法による近似文字列照合を組み合わせることによって行う．

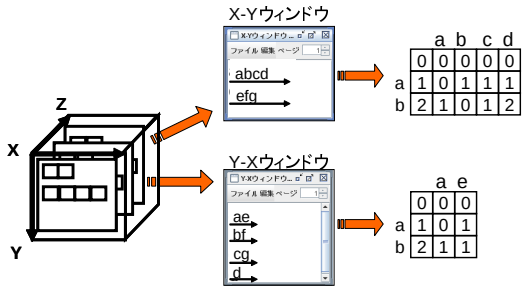

図 2.4: 平面ウィンドウ

照合の流れは図 4 の通りである．

まず 3 次元テキストエディタにおいて文字データを編集し，3 次元テキストエリア内に文字データを格納する．2 次元近似文字列照合を行うためには平面ウィンドウにより断面を選択し，格納された 3 次元文字データから文字面を取り出す．そして平面軸方向に並ぶ文字を行ごとに文字列として扱い，パターンと許容エラー数 k を定めた上で，動的計画法による近似照合を行う．平面軸方向とは例えば扱う平面が X-Y 平面であれば X 軸・Y 軸，といった具合である．しかし実際は 2 章の脚注（1）で述べたように，平面ウィンドウ上での文字データは 1 次元データである．例えば X-Y 平面ウィンドウにより 3 次元テキストエリア中の文字データが取り出されたならば，それは ”X 方向に展開した文字列の集合” が X-Y 平面ウィンドウに表示されることになる．この文字列集合における文字列一つ一つに対してパターンと動的計画法により近似照合を行えば，横軸方向文字列への近似照合ができる．これに加えて縦への近似照合を行うには，先ほどの平面ウィンドウに対する軸反転させた平面ウィンドウを用いればよい．この場合対応するのは Y-X 平面ウィンドウであり，X-Y 平面ウィンドウによって表示される文字データが横書きだとすると，縦書き表示となって現れる．つまりこ

のウィンドウで横軸方向への近似照合を行えば，結果として X-Y 平面ウィンドウでの縦軸方向への近似照合と同様の結果を得ることになる．

2.5 実験

最初に，実験で使用した文書データの詳細についてと今回の実験の評価方法について述べる．その後に実験を行い，得られた実験結果から考察を行う．

2.5.1 実験環境

今回の実験では文書データにロイターコーパス (Reuters-21578) を用いて実験する．このデータは全て SGML ファイルであるため，タグが存在する．だがタグによる記号が存在すると近似する文字列が少なくなってしまうと予想したため，この実験ではデータ中のタグを除去して実験を行うことにする．この文書データはファイルが計 22 個存在する．3 次元テキストエディタに対してこの文書ファイルを時系列に並べ，ページと見なして 3 次元文字データとし，実験を行う．

2.5.2 評価方法

第 4 章で述べたような 3 次元テキストエリア内の文字データから，Z=1,2,3,…としたときの各 X-Y 平面に対して X 方向・ Y 方向へのパターン近似文字列照合を行う．そのときに許容エラー数 k 以内のエラー数となる部分を動的計画法による近似文字列照合により探索し，パターン近似数，近似照合所要時間 [s] を算出する．しかし同様に Y-Z 平面や X-Z 平面などを扱えば Z 方向への近似文字列照合が可能であることも容易に理解できる．従って，X 方向・Y 方向・Z 方向の 3 次元へのパターン近似数，近似照合所要時間を算出することにする．パターン P については,

- abcd
- abcdefgh
- abcdefghijkl

の 3 種類を与え，それぞれの場合で許容エラー率 ϵ,

- 25%
- 50%
- 75%

と設定し，実験を行う許容エラー率 ϵ とは許容エラー数 k をパターンの文字列数で割った値である．つまり，パターン abcd のとき許容エラー率が 25%，50%，75%であるなら，許容エラー数はそれぞれ 1，2，3 となる．

2.5.3 実験結果

パターン P が abcd ，許容エラー率（許容エラー数 k）が $25\%(k=1)$，$50\%(k=2)$，$75\%(k=3)$ のときのパターン近似数，照合所要時間 [s] は表 1 の通りである．

表 2.1: パターンが abcd の場合

	X	Y	Z
$\epsilon = 25\%$			
(近似数)	195	1078	842
(時間 [s])	8.862	71.292	49.091
$\epsilon = 50\%$			
(近似数)	354976	129851	105183
(時間 [s])	8.932	71.376	49.154
$\epsilon = 75\%$			
(近似数)	5004756	5240420	4423556
(時間 [s])	9.213	71.950	49.377

次に，パターン P が abcdefgh ，許容エラー率（許容エラー数 k）が $25\%(k=2)$，$50\%(k=4)$，$75\%(k=6)$ のときのパターン近似数，照合所要時間 [s] は表 2 の通りである．

表 2.2: パターンが abcdefgh の場合

	X	Y	Z
$\epsilon = 25\%$			
(近似数)	0	0	0
(時間 [s])	15.048	109.477	66.135
$\epsilon = 50\%$			
(近似数)	4187	2146	1198
(時間 [s])	15.069	109.538	66.112
$\epsilon = 75\%$			
(近似数)	3503650	1763207	1308192
(時間 [s])	15.385	109.748	66.272

最後に，パターン P が abcdefghijkl ，許容エラー率（許容エラー数 k）が $25\%(k=4)$，$50\%(k=8)$，$75\%(k=12)$ のときのパターン近似数，照合所要時間 [s] は表 3 の通りである．

近似照合数についてはパターンがどの場合でも，許容エラー率が増えるほど増加する傾向が見てとれる．

2.5.4 考察

近似照合の際の照合所要時間について考察する．今回の照合法で表 5〜7 の結果からわかることは，X 方向の照合所要時間に対し Y・Z 方向のそれはかなり大きな値を

表 2.3: パターンが abcdefghijkl の場合

	X	Y	Z
$\epsilon = 25\%$			
(近似数)	0	0	0
(時間 [s])	21.264	148.660	83.675
$\epsilon = 50\%$			
(近似数)	140852	35533	19068
(時間 [s])	21.327	148.757	83.740
$\epsilon = 75\%$			
(近似数)	15648093	102510809	4737057
(時間 [s])	23.421	160.845	88.284

とっている，ということである．つまり許容エラー数（k）に関わらず，格納方式（どの軸で検索を行うか）に依存して負荷が発生している．これは当該エディタの実現手法によるものであり，可変配列を入れ子にした文字データ格納空間の構造上避けられない点であると考えられる．

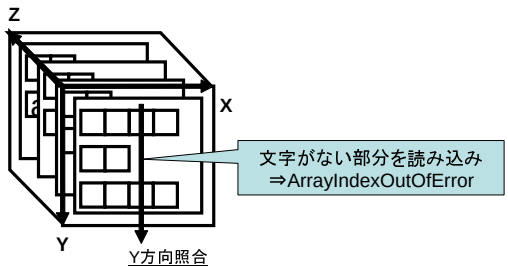

図 2.5: Y 方向への照合は要素の無い場所を読んできてしまう

例えば 3 次元テキストエリアから Y 方向に文字データを参照する場合，図 5 のように参照すると，文字が無い点を途中で参照してしまい，”ArrayIndexOutOfError”のエラーを引き起こす．従ってこのエラーを防ぐためには，今参照しようとしている 3 次元座標（X, Y, Z）に事前に文字があるか調べてから，文字を取り出す必要がある．Y・Z 方向への参照はこのチェックが必要になるため，結果として X 方向に比べて照合負荷が大きくなる．しかし，この負荷によるパターン照合数への影響などはなく，21,578 件（Z 軸上の深さ）の文字データから近似照合は実現できているため十分実用に耐える事ができると考えられる．

2.6 結論

本研究では動的計画法を用いた近似文字列照合を 2 次元の文字データ及び 3 次元の文字データに対して行う手法を提案した．結果，従来の 1 次元近似文字列照合では探索できない新たな次元方向への近似文字列照合を実現できた．今回は X/Y/Z 方向への 3 次元近似まで示したが，発展させれば 3 次元空間内の対角線方向での近似文字列照合も可能である．

第3章　遷移確率距離を用いた類似文字列検索

本研究では，文字列間距離をマルコフ過程による遷移確率で推定した類似検索法を提案する．文字列中の文字は確率過程的に出現すると仮定し，出現確率を非類似度として用いたDP 類似検索アルゴリズムを提案する．この手法は，バイオインフォマティクス分野のホモロジー検索アルゴリズムに対応しており，他の類似検索手法と比較することで有用性を評価する．

3.1　前書き

入力文字列をパターン，比較対象文字列（または比較対象文字列集合）をテキスト（テキスト集合）としたとき，パターンに類似した文字列をテキスト（テキスト集合）から探索することを類似文字列検索という．パターン・テキスト間の違い（距離）は文字列距離を計算することにより決定でき，編集距離・ハミング距離等の距離関数が存在する [1]．例えば“ $bath$ ”と“ $birth$ ”は“ a ”と“ ir ”が異なり，編集距離は 2 となる．これらの距離関数を用いれば，パターンに類似する文字列を探索できる．しかし，同じ距離の文字列が複数あるとき，これらの中で最も意図に合致するものを検出するには，文脈情報など他の情報を付加する必要があり，単純に類似検索を適用するわけにはいかない．

本研究では，文脈情報などを付加せずに「パターンの意図に最も類似する文字列の検索」を実現するため，文字列距離に確率的な概念を取り入れた遷移確率距離（Markovian Distance）による動的計画法（DP）類似検索アルゴリズムを提案する．遷移確率距離とは，文字列中の文字は，直前の部分文字列（q-gram）により確率過程的に出現すると仮定し，その確率が高い遷移にはコスト（距離）を低く，低い遷移にはコストを高くした距離関数である．これにより文字列中の文字の並びを確率的に正しいか判定し，尤もらしい文字の並びに訂正することが可能になる．例えば“ $boook$ ”において“ oo ”の後に“ o ”が続く確率は低いため，より確率が高い次の文字“ k ”に遷移し，結果として正しいスペル英単語“ $book$ ”に一致する．従来の距離関数では文字列距離が整数値になるために同距離の文字列が多く検出されてしまっていたが，確率（実数値）の高低差をコスト計算に用いる遷移確率距離により，各文字列の距離も実数値となり，パターンに最も類似する文字列の検索も容易になる．

本稿では，遷移確率距離による類似検索手法を述べ，編集距離を用いた類似検索手法と比較することで，有用性を示す．本手法ではDPアルゴリズムを扱う．DPを用いた類似検索手法は各テキストと厳密に距離計算を行うため，高速化が容易でない．しかし，事前に「明らかに類似しそうにない文字列」を排除することで，計算時間を減らすことができる．この考え方はバイオインフォマティクス分野 [4] のホモロジー検索（DNA・アミノ酸配列の類似検索）で用いられ，実際にBLASTアルゴリズム [5] 等に適用されている．ホモロジー検索で扱う DNA データは長大文字列であるが，本手法で扱う文字列は短く，BLAST アルゴリズムをそのまま適用して高速化させることは容易でない．ここでは，短い文字列に対応させた新たな高速類似検索のための「非類似文字列排除アルゴリズム」を提案する．遷移確率距離 DP 類似検索手法により，DP 計算の時間を短縮でき，高速化を実現することができる．本稿では，実験によりこのアルゴリズムの高速性を評価する．

本論文の構成は次の通りである．第 2 章で従来の文字列距離と遷移確率距離，そして遷移確率距離を用いた DP 類似検索手法について述べる．第 3 章では,「非類似文字列排除による類似検索の高速化」を述べる．第 4 章で実験とその結果による有用性の評価・考察を示し，最後に第 5 章で結論とする．

3.2 遷移確率距離とその DP 類似検索

入力文字列をパターン P ，比較対象文字列をテキスト T とするとき，P と T の類似度とは，その 2 文字列間の距離（エラーの度合い）と定義する．距離計算には最も一般的な手法として動的計画法（DP）を用いる．本手法でもそのDPによる計算法を適用して文字列の距離を計算する．

本章では，従来の編集距離（Edit Distance）とハミング距離（Hamming Distance），本稿で提案する遷移確率距離（Markovian Distance）について概説する．また遷移確率距離を用いたDP類似検索手法について述べる．

3.2.1 編集距離とハミング距離

パターン $P_{1...n} = p_1p_2...p_n$ と テキスト $T_{1...m} = t_1t_2...t_m$ の編集距離 $ed(P,T)$ は，P を T に一致させるための最小修正操作コストと定義する．修正操作には 1 文字の置換・削除・挿入があり，操作を 1 回行うごとにコストが 1 ，つまり距離が 1 増加する．例えば P を“ $zcde$ ”，T を“ $abcd$ ”とするとき，P と T の距離 $ed(P,T)$ を求めると,“ $zcde$ ”→“ $acde$ ”(置換) →“ $abcde$ ”(挿入) →“ $abcd$ ”(削除) の操作手順により P を T に一致させることができ，$ed(P,T)$ は 3 となる．

一方，ハミング距離は置換操作のみしか許されない距離関数である．例えば 2 文字列のハミング距離 $hd(P,T)$ を求めると,“ $zcde$ ”→“ $acde$ ”→“ $abde$ ”→“ $abce$ ”→“ $abcd$ ”というように全ての文字に置換操作が行われ，$hd(P,T)$ は 4 となる．

3.2.2 遷移確率距離

本稿では遷移確率距離を提案する．パターン $P_{1...n} = p_1 p_2 ... p_n$ とテキスト $T_{1...m} = t_1 t_2 ... t_m$ が与えられたとき，この 2 文字列の遷移確率距離を $md(P,T)$ と表す．P を T に一致させるための最小修正コストや，修正操作に置換・削除・挿入を用いることは編集距離と同じである．しかし，遷移確率距離では操作回数を考えるのではなく，操作対象となる文字に対する q-gram 遷移確率を用いる．P を T に一致させるために，P の位置 i で修正操作が行われ，本来後続にある文字 p_i は文字 z に変化するとする．従って，その位置まで続いていた部分文字列に対する q-gram 遷移確率も変化する．その変化をコスト化し，遷移確率距離とする．

P が T に修正されるとき， P 内の文字を調べて T に変える．文字 p_i を調べるとき， p_i まで続く文字列は T の部分文字列であり，p_i が z に変化すると，その q-gram 遷移確率は $P(p_i|t_{j-q+1}...t_{j-1})$ から $P(z|t_{j-q+1}...t_{j-1})$ に変化する．ここで，j は T の位置を表す．即ち， T の部分文字列用いて q-gram 遷移確率に基づく距離計算をする．q-gram 遷移確率情報は文書コーパスを元に算出する．コーパス中に出現する q-gram の頻度を基に確率値を求める．

置換

パターン P をテキスト T に一致させるために，P 内の文字 p_i を T 内の文字 t_j に置換する．p_i までの q-gram の遷移確率は $P(p_i|t_{j-q+1}...t_{j-1})$ ，t_j までの q-gram の遷移確率は $P(t_j|t_{j-q+1}...t_{j-1})$ である．置換コスト δ_{rep} は以下の式から求められる．

$$\delta_{rep} = \frac{\ln P(t_j|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})} \tag{3.1}$$

$P(t_j|t_{j-q+1}...t_{j-1})$ の方が大きい場合，置換を行うことで「尤もらしい文字の並び」になったと考えられ，置換コストは低いものとなる．一方 $P(p_i|t_{j-q+1}...t_{j-1})$ の方が大きい場合,「生じにくい文字の並び」に近づくと考えられ，置換コストは高くなる．図 4.1 は“ $solt$ ”内で“ o ”から“ t ”へ遷移するところを置換操作により“ d ”への遷移に変更した場合を示す．3-gram 遷移確率を利用するとき，遷移文字変更により $P(t|ol)$ から $P(d|ol)$ に変化する．

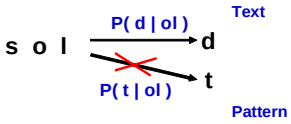

図 3.1: 遷移確率距離における置換操作

以下で述べる挿入・削除コストの式も同様に，操作前の確率が分子，操作後の確率が分母の形をしている．従って操作前の確率が低ければ操作コストは大きくなり，操作後の確率が高ければ操作コストは小さくなる．

挿入

パターン P に生じる文字 p_i の前にテキスト T にある文字 t_j を挿入する．挿入前に続くときの文字 p_i までの q-gram の出現確率は $P(p_i|t_{j-q+1}...t_{j-1})$ であり，挿入により後続文字として生じる文字 t_j までの q-gram の出現確率は $P(t_j|t_{j-q+1}...t_{j-1})$ である．挿入コスト δ_{ins} は以下の式で求められる．

$$\delta_{ins} = \frac{\ln P(t_j|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})} \tag{3.2}$$

図 4.2 は“ $plan$ ”で“ a ”と“ n ”の間に文字“ i ”を挿入し，遷移を変更する場合を示す．3-gram 遷移確率の場合，その確率は $P(n|la)$ から $P(i|la)$ に変化する．

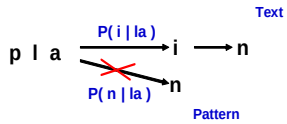

図 3.2: 遷移確率距離における挿入操作

対象となる文字と出現確率の形が変わらないため置換の式と同様の形となっている．

削除

パターン P 内の文字 p_i を削除することで，後続文字は p_i から p_{i+1} に変化する．従って，p_i までの q-gram 出現確率 $P(p_i|t_{j-q+1}...t_{j-1})$ ，p_{i+1} までの q-gram 出現確率 $P(p_{i+1}|t_{j-q+1}...t_{j-1})$ を用いて，削除コスト δ_{del} を以下の式で定義する．

$$\delta_{del} = \frac{\ln P(p_{i+1}|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})} \tag{3.3}$$

図 4.3 は“ $plain$ ”で“ a ”と“ n ”の間に遷移の文字“ i ”を削除し，“ a ”から“ n ”に遷移する場合を示す．

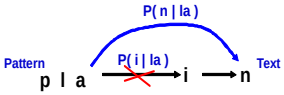

図 3.3: 遷移確率距離における削除操作

3.2.3 遷移確率距離を用いた DP 類似検索

DP により，パターン $P_{1...n}$ とテキスト $T_{1...m}$ における距離を遷移確率距離を用いて計算するアルゴリズムを示す [1]．列に P ，行に T を配置した行列 $C_{0...m,0...n}$ を作成し，その行列に以下の 3 つの式に基づいて値を埋めることにより P・T 間の遷移確率距離を求める．

$$C_{0,0} = 0 \tag{3.4}$$

$$C_{j,0} = C_{j-1,0} + \frac{\ln P(t_j|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})} \tag{3.5}$$

$$C_{0,i} = C_{0,i-1} + \frac{\ln P(p_{i+1}|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})} \tag{3.6}$$

$$C_{j,i} = min \begin{cases} C_{j-1,i-1} + \delta(p_i, t_j) \\ C_{j-1,i} + \frac{\ln P(t_j|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})} \\ C_{j,i-1} + \frac{\ln P(p_{i+1}|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})} \end{cases} \tag{3.7}$$

(4.4) 式は初期値であり，距離 $C_{0,0}$ は 0 である．(4.5) 式は最上行の行列値に対する式であり，左に位置する行列値 $C_{j-1,0}$ に t_i を挿入する追加コストを表す．(4.6) 式は最左列にある行列値に対する式であり，上に位置する行列値 $C_{0,i-1}$ に p_j を削除する追加コストを表す．(4.7) 式は行列における左上・左・上の値があらかじめ計算されているとき，それらの値に操作コストを加えることによって $C_{j,i}$ が求まることを表す．$C_{j-1,i-1}$ から遷移した場合，比較文字列に文字 p_iと t_j が追加される．p_iと t_j が一致している場合，$\delta(p_i, t_j) = 0$ となる．一致していない場合，$\delta(p_i, t_j)$ は置換コストとなり，$\delta(p_i, t_j) = \frac{\ln P(t_j|t_{j-q+1}...t_{j-1})}{\ln P(p_i|t_{j-q+1}...t_{j-1})}$ となる．$C_{j-1,i}$ から遷移した場合は p_i が削除されていることを表し，$C_{j-1,i}$ に削除コストが追加される．$C_{j,i-1}$ から遷移した場合は t_j が挿入されていること表し，$C_{j,i-1}$ に挿入コストが追加される．これらの中で，コスト和が最小となる値を $C_{j,i}$ とする．

計算によって得られた行列値 $C_{i,j}$ は $P_{1...i}$ と $T_{1...j}$ の距離を表している．従って行・列が共に末尾である行列値 $C_{m,n}$ が P・T 間の距離を示す．

本稿では遷移確率距離を扱うが，各修正操作で扱う q-gram 遷移確率を 1 とすれば編集距離になる．第 2 章に示したパターン“ $zcde$ ”，テキスト“ $abcd$ ”の DP 計算の様子を図 4.4 に示す．この図は編集距離による DP 計算を示す．

この DP 計算法によるパターン類似度計算をテキスト集合に用いることで，類似検索の実現が可能になる．集合内の各テキスト T に対してそれぞれ DP 計算を行い，パターン P との距離を調べる．その中で最も類似する，あるいは上位であるテキスト T をパターン P と類似すると見なせる．

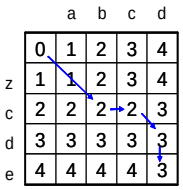

図 3.4: DP による編集距離計算

3.3 非類似文字列排除による類似検索の高速化

DP 類似検索は効果的だが，計算時間が膨大になることが知られている [1]．DP 計算により，パターン P とテキスト T 間の距離を厳密に調べ，明らかに類似しそうもない文字列を事前に排除して，残りの文字列に対して DP 計算を行う方が効率的である．類似度の高い領域を見つけ，そのヒットした類似領域数の割合が閾値より高いものだけ DP を適用すれば，明らかに類似していないテキスト T は DP 計算の対象から排除でき，高速化が期待できる．

部分的な領域で類似する箇所を見つけ出し，その後閾値 V により DP 計算を行うべき文字列かどうかの判定を行う手法は，バイオインフォマティクス分野で扱われる BLAST アルゴリズム（ホモロジー検索手法）でも利用されている．DNA やタンパク質は 1 次元の文字列データとして扱うことができるが，これらのデータは長大文字列であることが多い．そのために DP 計算の時間をできるだけ減らし，高速化を実現するというのが BLAST アルゴリズムの大まかな流れである．本稿で扱う文字列は英単語等の短い文字列であり，このアルゴリズムをそのまま用いると返って複雑化してしまうため，より容易に非類似文字列を排除するアルゴリズムを提案する．

提案手法の流れは以下の通りである．

1. パターン P を固定長に分割して正規表現文字列生成
2. テキスト T から正規表現文字列に一致する箇所を検索
3. 一致箇所数の割合が高いテキスト T に対して DP 計算

手順（1）では，入力文字列のパターン P を長さ l の固定長文字列 W に分割する（以下，分割文字列 W とする）．分割文字列 W の総数を s とすると，$P = W_1W_2...W_s$ となる．さらにその分割文字列集合 $\{W_1, W_2,..., W_s\}$ に対してハミング距離 hd 以内の正規表現文字列 $\{R_1, R_2,..., R_s\}$ を生成する．手順（2）では，テキスト T 内で各正規表現文字列 R と一致する箇所を検索する．検索の制約として，k 番目の正規表現文字列 R_k の一致する部分は $k-1$ 番目の正規表現文字列 R_{k-1} よりも後ろの位置とする．R_{k-1} がテキスト T の位置 j から開始していた場合，R_k の探索範囲は位置 $j+1$ 以降となる．さらに R_k が位置 j'（$j < j'$）から開始していた場合，R_{k+1} の探索範囲は位置 $j'+1$ 以降に続く．しかし， R_k が T で一致しないとき，R_{k+1} の探索範囲は

R_k 同様の位置 $j+1$ 以降とする．手順（3）では，一致箇所数の割合を調べ，高いテキスト T に対してのみ遷移確率距離 DP 計算を行う．分割文字列 W の総数 s に対する正規表現文字列 R のテキスト T 内での一致数の割合をヒット率 H と定義する．H に対して閾値 V を設定し，V 以上のヒット率であるテキスト T に対してのみ遷移確率距離を用いた DP 計算を行う．

例えば P を“ *thuuner* ”，T を“ *thunder* ”とする．P から長さ 2 の分割文字列 W を生成すると，“ *th* ”，“ *uu* ”，“ *ne* ”，“ *r* ”となる（順に W_1, W_2, W_3, W_4 ）．このとき，W_1 からハミング距離 1 の正規表現文字列 R_1 は，$\{t.\ ,\ .h\}$ の 2 種類生成できる（ピリオドは正規表現で任意の 1 文字を表す）．他の正規表現文字列 R_2 ，R_3 ，R_4 についても同様に 2 種類ずつ生成できる．次にテキストからの検索を W_1 から順に行う．R_1 の場合，どちらの正規表現文字列もテキストの先頭位置から一致する．従って $R_2 = \{u.\ ,\ .u\}$ は 2 番目の文字以降で一致位置を検索する．“ $u.$ ”はテキスト内 3 番目の文字位置から一致するが，“ $.u$ ”は 2 番目位置から一致するためそちらが適用され，$R_3 = \{n.\ ,\ .e\}$ は 3 番目の位置から検索を行う．R_3 ，R_4 について同様に一致箇所検索を行った結果，図 3.5 上部のように，正規表現文字列に対して一致箇所が存在し，ヒット率 H は 100%（分割文字列総数 4 に対して一致箇所数 4）となる．テキストが“ *however* ”で同様の一致箇所検索を行う場合，図 3.5 下部のように一致箇所が 2 つ存在する．従ってそのヒット率 H は 50%（分割文字列総数 4 に対して一致箇所数 2）となる．

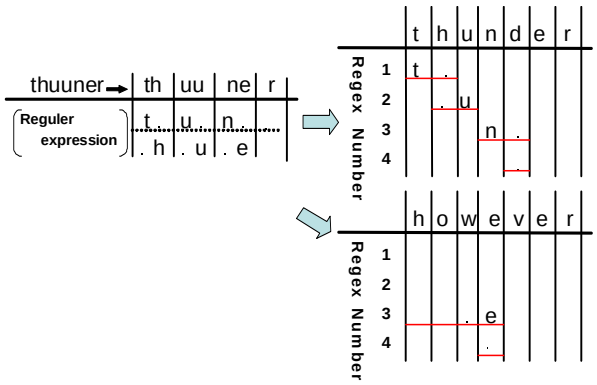

図 3.5: パターン“ *thuuner* ”，テキスト“ *thunder* ”“ *however* ”のときの非類似文字列排除アルゴリズム

3.4 実験

遷移確率距離による類似検索手法を用いて「スペルミス英単語の訂正」を目的とする実験を行う．パターンとしてスペルミスした英単語を，テキスト集合として辞書コー

パスから抽出した単語集合を与え，遷移確率距離を用いた DP 類似検索により性能を測定する．

まず，実験で使用した文書データの詳細と評価方法について述べ,「スペルミス英単語の訂正」を行い，その結果から考察を行う．

3.4.1 実験環境

初めに，遷移確率距離計算に用いる 3-gram 遷移確率情報をロイターコーパス（Reuters-21578）から得る．このコーパスは SGML 形式の 21,578 件のロイター新聞記事データである．SGML タグ部分を事前に除去し，すべて小文字に変換してからその遷移確率を計算する．その結果，ロイターコーパス内には 58 個の文字・記号が存在し（表 4.1），3-gram 遷移確率情報を求める．全体として 58^3 個の確率値があり，表 3.2 では一部分を示す．

表 3.1: ロイターコーパス内に存在する文字

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7
8	9	0		!	”	$	’	(	)	*	+	,	-	.	/	:
;	=	?	[	]	^	_	~									

表 3.2: 遷移確率値

	a	b	c	d	e	f
aa	0.086	0.033	0.022	0.009	0.002	0.011
ab	0.017	0.004	0.004	0.004	0.016	0.000
ac	0.007	0.000	0.144	0.000	0.085	0.000
ad	0.040	0.006	0.004	0.116	0.271	0.000
ae	0.005	0.002	0.051	0.015	0.001	0.005
af	0.006	0.001	0.001	0.000	0.034	0.246
ag	0.168	0.000	0.000	0.000	0.396	0.000
ah	0.101	0.002	0.002	0.014	0.279	0.001
ai	0.000	0.000	0.001	0.653	0.000	0.000
aj	0.006	0.000	0.000	0.000	0.011	0.000
ak	0.053	0.001	0.000	0.006	0.665	0.001
al	0.022	0.002	0.005	0.005	0.085	0.012
am	0.061	0.039	0.009	0.002	0.416	0.001

類似検索対象となるテキスト集合として，英辞郎辞書の見出し語を用いる．ロイターコーパス内の 58 個以外の文字・記号に対しては遷移確率が存在せず，距離計算が行えないため，それらの文字が含まれる単語は除外する．この結果，英辞郎辞書コーパスから抽出する単語数は 294,642 個となる．

3.4.2 実験方法 1

パターン P （スペルミス文字列）に類似する単語をテキスト集合（英辞郎コーパス単語）内から検索する．従来手法である編集距離 DP 類似検索，本手法で提案する遷移確率距離 DP 類似検索，非類似文字列排除アルゴリズムを用いた遷移確率距離 DP 類似検索の 3 手法についてそれぞれ性能を比較する．パターン P には，ロイターコーパス頻出単語上位 50 個から無作為に選出した単語に対し，乱数でスペルミスを起こした文字列をパターン P に用いる．本実験では“ $general$ ”,“ $under$ ”の 2 単語を選出し，それぞれに対し編集距離 1 のスペルミスの語“ $geneeral$ ”,“ $undr$ ”，編集距離 2 のスペルミスの語“ $deneraol$ ”,“ $aner$ ”の 4 種類の文字列をパターン P に用いる．

遷移確率距離 DP 検索では，パターン P を長さ l の分割文字列 W に分割し，W に対するハミング距離 hd の正規表現文字列 R 集合のテキストに対するヒット率 H を調べ，閾値 V 未満であるものに関しては DP 計算対象から除外する．本実験では $l = 2$, $hd = 1$ として，各パターンに V を 2 種類設ける．“ $geneeral$ ”,“ $deneraol$ ”(分割文字列数 $s = 4$）の場合は $V = 75\%, 100\%$ ，即ちテキストに一致する分割文字列 W の数が 3 つ以上，あるいは 4 つ全てであるとき,“ undr ”,“ $aner$ ”(分割文字列数 $s = 2$）の場合は $V = 50\%, 100\%$，即ちテキストに一致する分割文字列数 W が 1 つ以上，あるいは 2 つ全てであるとき DP 計算を行う．

本実験では，各手法に対する「最小文字列距離 3 単語とその文字列距離」,「実行時間」を調べて比較する．比較評価については「遷移確率距離 DP に対する実行時間の割合」を算出して行う．また，非類似度文字列排除アルゴリズムを用いた手法では DP 計算を行う単語の数が減少するため，各手法についての「 DP 計算単語数」を調査し，比較を行う．

3.4.3 実験方法 2

各類似検索手法に対する検索精度の評価実験を行う．スペルミス文字列を与えたパターン P に類似した単語を，正しい単語の集合であるテキスト集合から検索することは実験方法 1 と同様である．実験方法 2 ではロイターコーパス頻出単語上位 50 個すべてに編集距離 1 のスペルミスを乱数発生させ，それぞれをパターンに用いて類似検索を行う．検索結果上位 k 個以内に正しい単語が含まれているときを正解とし，全 50 個の文字列に対する正解の数の割合を「正解率」とする．$k = 1$, $k = 3$ のときの場合で正解率を算出する．正解単語と同じ距離の単語が複数出現し，最上位からの単語数の総和が k 個以上になる場合，正解単語を絞り込めていないと判断し，正解とは見なさない．例えば $k = 3$ で検索結果同率 1 位の単語が 4 個存在し，その中に正解単語が含まれる場合，k を超える単語数であるため，正解とは見なさない.「編集距離 DP」,「遷移確率距離 DP」,「非類似文字列排除アルゴリズムを用いた遷移確率距離 DP」の 3 手法で実験を行い，正解率により比較評価を行う．非類似文字列排除アルゴリズムを用いた遷移確率距離 DP 類似検索で分割文字列長 l，ハミング距離 hd は実験方法 1

と同じとする．しかしヒット率閾値 V はパターン文字列関係なく 100% として実験を行う．

3.4.4 実験方法 1 に対する実験結果

各手法に対する「最小文字列距離 3 単語とその文字列距離」をパターン“ *geneeral* ”, “ *deneraol* ”, “ *undr* ”, “ *aner* ”ごとに表 3.3，3.4，3.5，3.6 に示す．左上の表が編集距離 DP，右上が遷移確率距離 DP，下の表が非類似文字列排除手法を用いた遷移確率距離 DP の検索結果を表す．

表 3.3: パターン“ *geneeral* ”のとき

単語順位	全単語編集 DP	
	類似単語	編集距離
1	general	1.0
2	genera	2.0
〃	genecial	2.0

単語順位	全単語遷移 DP	
	類似単語	遷移確率距離
1	general	0.691
2	genecial	1.724
3	parenteral	1.426

単語順位	非類似単語遷移 DP（3/4）		非類似単語遷移 DP（4/4）	
	類似単語	遷移確率距離	類似単語	遷移確率距離
1	general	0.691	general	0.691
2	genecial	1.724	genecial	1.724
3	parenteral	1.426	parenteral	1.426

表 3.4: パターン“ *deneraol* ”のとき

単語順位	全単語編集 DP	
	類似単語	編集距離
1	demerol	2.0
〃	general	2.0
3	central	3.0
〃	deaerate	3.0
〃	以下距離 3.0 の単語 43 個	

単語順位	全単語遷移 DP	
	類似単語	遷移確率距離
1	general	1.284
2	generate	1.379
3	generation	1.505

単語順位	非類似単語遷移 DP（3/4）		非類似単語遷移 DP（4/4）	
	類似単語	遷移確率距離	類似単語	遷移確率距離
1	general	1.284	general	1.284
2	generate	1.379	central	1.586
3	generation	1.505	generable	1.744

表 3.5: パターン“ *undr* ”のとき

単語順位	全単語編集 DP	
	類似単語	編集距離
1	under	1.0
〃	undo	1.0
2	and	2.0
〃	anda	2.0
〃	以下距離 2.0 の単語 110 個	

単語順位	全単語遷移 DP	
	類似単語	遷移確率距離
1	under	0.382
2	used	0.566
3	utner	0.615

単語順位	非類似単語遷移 DP（1/2）		非類似単語遷移 DP（2/2）	
	類似単語	遷移確率距離	類似単語	遷移確率距離
1	under	0.382	under	0.382
2	used	0.566	utner	0.615
3	utner	0.615	user	0.666

表 3.6: パターン“ *aner* ”のとき

単語順位	全単語編集 DP	
	類似単語	編集距離
1	abner	1.0
〃	acer	1.0
〃	ager	1.0
〃	以下距離 1.0 の単語 14 個	

単語順位	全単語遷移 DP	
	類似単語	遷移確率距離
1	and	0.353
2	saner	0.356
3	maner	0.377

単語順位	非類似単語遷移 DP（1/2）		非類似単語遷移 DP（2/2）	
	類似単語	遷移確率距離	類似単語	遷移確率距離
1	and	0.353	saner	0.356
2	saner	0.356	maner	0.377
3	maner	0.377	caner	0.52

パターン“ *geneeral* ”（表 3.3），“ *deneraol* ”（表 3.4），“ *undr* ”（表 3.5）では編集距離 DP，遷移確率距離 DP のどちらの手法を用いた場合でも“ *general* ”，“ *under* ”を最も類似した単語として正しく検出している．しかし編集距離 DP では，“ *deneraol* ”，“ *undr* ”，また“ *aner* ”（表 3.6）に対して同率順位（同じ距離）の単語を複数検出している．例えばパターン“ *deneraol* ”において，編集距離 DP では最上位類似単語として“ *demerol* ”，“ *general* ”の 2 単語を検出しており，曖昧な検索結果となっている．一方，遷移確率距離 DP では各単語に対しての距離を差別化し，最上位類似単語として“ *general* ”（距離 1.284）のみを検出しているため，その結果をそのままスペル訂正単語として反映できる．

次に各手法での「実行時間」,「DP 計算単語数」,「遷移確率距離 DP に対する実行時間の割合（時間率）」を表 3.7，3.8，3.9，3.10 にパターンごとに示す．各パターンにおいて，ヒット率閾値 100% では，全単語に対する遷移確率距離 DP よりも高速である．しかし，パターン“ *aner* ”では，ヒット率閾値 50%の場合では余計に時間を要している．対象となる単語数は 262,733 個であり，排除した単語数が少ない．このため排除手法に対する実行時間も要するため，排除した単語数が少ない場合，この手法を用いていない時（34.780 秒）よりも多くの時間（37.694 秒）を要する．高速化を実現するためにはヒット率の閾値を大きく取る必要がある．

表 3.7: パターン“ *geneeral* ”のとき

手法	実行時間 [s]	DP 単語数	時間率 [%]
全単語編集 DP	4.079	294,462	7.25
全単語遷移 DP	56.301	294.462	100.0
非類似遷移 DP（75%）	36.663	97,602	65.12
非類似遷移 DP（100%）	14.842	15,764	26.36

表 3.8: パターン“ *deneraol* ”のとき

手法	実行時間 [s]	DP 単語数	時間率 [%]
全単語編集 DP	3.945	294,462	6.86
全単語遷移 DP	57.543	294.462	100.0
非類似遷移 DP（75%）	52.906	185,339	91.94
非類似遷移 DP（100%）	14.611	13,130	25.39

表 3.9: パターン“ *undr* ”のとき

手法	実行時間 [s]	DP 単語数	時間率 [%]
全単語編集 DP	2.454	294,462	6.78
全単語遷移 DP	36.192	294.462	100.0
非類似遷移 DP（50%）	37.954	246,140	104.9
非類似遷移 DP（100%）	17.595	59,840	48.62

表 3.10: パターン“ *aner* ”のとき

手法	実行時間 [s]	DP 単語数	時間率 [%]
全単語編集 DP	2.574	294,462	7.918
全単語遷移 DP	34.780	294.462	100.0
非類似遷移 DP（50%）	37.694	262,733	108.4
非類似遷移 DP（100%）	18.907	77,675	54.362

3.4.5 実験方法 2 に対する実験結果

各類似検索手法における正解率を表 3.11 に示す．編集距離 DP よりも遷移確率距離 DP の方が高い正解率を得ている．$k = 1$ の場合，編集距離 DP では 26.0 % であるが，最上位類似単語を複数検出していることが要因である．全パターン 50 個中，複数の単語を最上位単語として検出しているものが 36 個存在する．1 個が他の正しい単語に完全一致してしまい，正解単語を得られたものは 13 個存在するため，$13/50 = 26.0\%$ という結果となる．一方，遷移確率距離 DP では，単純に正解単語が最上位類似単語に現れているかどうかを示している．同様に $k = 1$ の場合，正解単語が得られたのは 32 個であるため，$32/50 = 64.0\%$ となる．非類似文字列排除手法を用いた場合では，正解単語数を排除した例が存在し，非類似文字列排除手法を用いない場合に比べて正解率は低下しているが，編集距離に比べて高い正解率を得る．

表 3.11: 各類似検索手法における正解率

検索手法	正解率（$k = 1$）[%]	正解率（$k = 3$）[%]
編集 DP	26.0	46.0
遷移 DP（全単語）	64.0	76.0
遷移 DP（非類似）	62.0	72.0

3.4.6 考察

編集距離 DP に対して，遷移確率距離 DP の類似検索は実行時間を要している．表 3.7，3.8，3.9，3.10 から実験に用いた各パターンにおける全単語編集距離 DP に対する全単語遷移確率距離 DP の比を示した表を表 3.12 に示す．編集距離 DP に対して遷移確率距離 DP の方が約 14 倍の実行時間を要している．

表 3.12: 編集距離・遷移確率距離での実行時間比

パターン	編集 [s]	遷移確率 [ns]	所要時間比
geneeral	4.079	56.301	13.80
deneraol	3.945	57.543	14.59
undr	2.454	36.192	14.75
aner	2.574	34.780	13.51

この原因は，DP におけるコスト計算にある．編集距離 DP ではパターン ，テキスト間での類似度計算において編集操作（置換，挿入，削除）コストは全て 1 である．一方，遷移確率距離 DP ではそのコストに遷移確率情報による実数値を用い，対数を取って分数計算を行っている．その違いが実行時間の違いにつながっている．具体例

として，DP 計算上で文字列“ ab ”の後続に“ c ”が存在するとき，“ d ”への置換操作を行う場合を考える．編集距離 DP では 1 が加わり，遷移確率距離 DP では $\frac{\ln P(d|ab)}{\ln P(c|ab)}$ の計算値が加わる．この置換コストの処理を各々 1000 万回繰り返した結果，前者は 180[ms]，後者は 2844[ms] の処理時間である．遷移確率距離 DP の置換コスト計算の方が 15.86 倍の処理時間である．しかし，編集距離 DP の追加コスト 1 を $\frac{\ln(1.0)}{\ln(1.0)}$ として同様に 1000 万回 繰り返した結果，1112 [ms] の処理時間である ．遷移確率距離 DP の置換コスト計算は 2.56 倍となり，先程の時間比よりも小さい値である．この時間比の差から，実数値の対数と分数計算を行ったことが大きく時間を要する原因であることがわかる．

しかし，遷移確率距離 DP では実行時間を要しているが，編集距離 DP に比べ高い正解率である．編集距離 DP に対し，遷移確率距離 DP の正解率が $k = 1$ の場合では約 40% ，$k = 3$ の場合では約 30 % 上昇している．この結果からスペル訂正を目的とする類似検索手法として，遷移確率距離 DP は有用であると評価できる．

遷移確率距離 DP は非類似文字列排除手法により高速化しているが，短いパターンで閾値が小さい値の場合では，その手法を用いないときの方が実行時間が短い．実験方法 1 のパターン“ undr ”,“ aner ”で閾値 50% の場合，非類似文字列排除手法を用いない遷移確率距離 DP に対する実行時間の割合が 104.9%，108.4% という結果となっている．この手法をより高速化するためには「文字列長の差」を非類似文字列排除手法で考慮する必要がある．パターンがより短い文字列，テキストの単語がより長大文字列である場合，その文字列長の差は距離に反映して値が大きくなる．しかし，同時にパターンに対するヒット率が高くなり，閾値以上となりやすい．例えば，英辞郎コーパスから抽出した単語には“ $hippopotomonstrosesquipedalian$ ”のような長大文字列の単語も存在する．パターンが“ $undr$ ”である場合，その分割文字列から生成した正規表現文字列の中に“ $.n$ ”,“ $.r$ ”が存在するが,“ $hippopotomonstrosesquipedalian$ ”に一致する箇所が存在する（“ on ”と“ tr ”）．しかし，この 2 文字列間の遷移確率距離は 18.115 となり，表 3.5 の“ $under$ ”(距離 0.382 ）と比べ，類似しているとは考えられない．このことから一定の文字列差がある場合，あらかじめ排除することで，高速化できる．

3.5 結論

本研究では確率的な文字列距離「遷移確率距離」を用い，DP 類似検索手法を提案した．従来の距離関数では複数の文字列が類似文字列として検出されるのに対し，遷移確率距離を用いた DP 類似検索では文脈情報を利用することなく，最も類似している文字列を検出できる．主たる応用として「スペルミス英単語訂正システム」が実現できる．スペルミス単語をパターン，テキスト集合に辞書コーパス単語を用いることでパターンに類似する単語を辞書コーパス単語内から訂正のために検索すればよい．

更に，事前に，明らかに類似していない単語を排除し，DP 計算を省く「非類似文字列排除アルゴリズム」を提案し，結果として DP 類似検索の高速化を実現した．

本手法を用いれば，ホモロジー検索のため，アミノ酸配列や DNA 配列内の「文字同士での置換の起こりやすさ」を指標とした類似検索ではなく,「後続文字の遷移しやすさ」による確率的類似検索として応用できる可能性がある.

第4章　動的確率距離による類似文字列検索

本研究ではマルコフ過程による確率を用いた類似文字列検索において動的に距離を更新する手法を提案する．文書は日々内容が変化し，その影響は文書内の文字の出現確率にも現れる．その影響を捉えた動的な確率距離によって，現在の情報を的確に反映した文字列間の類似度を示すことが可能になる．全体文書で得た統計的な確率距離と本研究で提案する差分更新型確率距離で類似度比較を行うことでその有用性を評価する．

4.1　前書き

類似文字列検索とは質問文字列（パターン）に対して類似度の高い文字列を文字列集合から抽出することである．文字列類似度は文字列距離によって計算でき，編集距離・ハミング距離等がある [1]．しかし，これらの距離では同じ距離の文字列が複数検索される場合がある．例えば，文字列“ $worh$ ”をパターンとして類似検索を行う場合，“ $word$ ”,“ $work$ ”,“ $worm$ ”等を英単語集合から得ることができる（編集距離 1）．この中で“ $worh$ ”に最も類似する文字列，つまりユーザーの意図に最も合致した文字列の検索には文脈情報等の他の情報の付加が必要不可欠である．

著者らは，文脈情報ではなく文字列中の文字の遷移確率を考慮する距離関数（遷移確率距離）により，意図に合致する文字列を検索する手法を提案した [8]．文字列中の文字が直前の部分文字列により確率過程的に出現すると仮定し，その確率が高い遷移にはコスト（距離）を低く，低い遷移にはコストを高くする距離関数である．この距離を用いた動的計画法（DP）類似検索アルゴリズムにより，文字列中の文字の並びを確率的に正しいか判定し，最もらしい文字の並びへの訂正が可能になる．[8] の実験結果から，編集距離に比べて約 30〜38% の精度の改善が得られたことを示している．

[8] の関連研究として，Wei が提案する Markov Edit Distance がある [6]．マルコフ確率場（Markov Random Field）を用いて，ある位置のデータが隣接地に存在するデータから影響を受けるというマルコフ確率的概念を加えた，拡張型編集距離を定義している．編集距離では本来文字列同士の比較において，文字の 1 対 1 の編集操作を想定している．しかし，Markov Edit Distance では多対多の編集操作を考え，それにより隣接データの影響を受けることで距離が決定する．この手法に対し，[8] ではコストを

直接確率値から定義して計算を行うため，コストの意味が明確になり，より判定しやすい結果を生む．

本研究では，[8] で著者らの提案した遷移確率距離を時刻に応じて動的に更新させる手法を提案する．ニュース記事等は日々内容が変化し，それに応じて記事内の文字遷移確率も変化する．従って，与えたパターンが同じであっても，時刻が異なればユーザーの意図に最も合致する検索文字列は異なる可能性がある．例えば，ユーザーの持つパターンがエラー文字列“ $worh$ ”である場合，時刻 t_1 で雇用関連の記事が多く出現したとすると，ユーザーは“ $work$ ”を最も意図していた可能性がある．一方，時刻 t_2 でコンピュータウィルス関連記事が増えた場合，最も意図するものが先ほどと異なり“ $worm$ ”になる可能性がある．このような意図する類似検索解の変化を，時系列文書における出現文字の変化から得ることによって，その時刻に応じた類似検索解の自動推定を実現する．

本稿の構成は次の通りである．第 2 章で遷移確率距離と動的計画法（DP）を用いた類似検索手法について述べる．第 3 章で本稿の目的である遷移確率距離の動的更新について論じる．第 4 章で実験とその結果に対する評価・考察を行い，最後に第 5 章で結論とする．

4.2 遷移確率距離

[8] で著者らが提案した遷移確率距離について説明する．まず，従来の距離関数について述べ，遷移確率距離と動的計画法（DP）を用いた類似検索手法を概説する．

4.2.1 編集距離とハミング距離

パターン $P_{1...l} = p_1p_2...p_l$ と テキスト $T_{1...m} = t_1t_2...t_m$ の編集距離 $ed(P,T)$ は，P を T に一致させるための最小修正操作コストと定義する．修正操作には 1 文字の置換・削除・挿入があり，操作を 1 回行うごとにコストが 1 増加する．例えば P を“ $zcde$ ”，T を“ $abcd$ ” とするとき，P と T の距離 $ed(P,T)$ を求めると，“ $zcde$ ”→“ $acde$ ”(置換) →“ $abcde$ ”(挿入) →“ $abcd$ ”(削除) の操作手順により P を T に一致させることができ，$ed(P,T)$ は 3 となる．

一方，ハミング距離は置換操作のみしか許されない距離関数であり，同様にハミング距離 $hd(P,T)$ を求めると，“ $zcde$ ”→“ $acde$ ”→“ $abde$ ”→“ $abce$ ”→“ $abcd$ ”となり，$hd(P,T)$ は 4 となる．

4.2.2 遷移確率距離

パターン $P_{1...l} = p_1p_2...p_l$ とテキスト $T_{1...m} = t_1t_2...t_m$ が与えられたとき，この 2 文字列の遷移確率距離を $md(P,T)$ と表す．P を T に一致させるための最小修正コス

トであること，修正操作に置換・削除・挿入を用いることは編集距離と同じである．しかし，遷移確率距離ではコスト計算に対して操作回数ではなく，操作対象となる文字に対する n-gram 遷移確率を用いる．P を T に一致させるために， P の位置 i で修正操作が行われ，本来後続にある文字 p_i は文字 c に変化するとする．従って，その位置まで続いていた部分文字列に対する n-gram 遷移確率も変化する．その変化をコスト化し，遷移確率距離とする．

P を T に修正するとき， P 内の文字を順次修正し，T に一致させる．文字 p_i を調べるとき， p_i まで続く文字列は T の部分文字列であり，p_i が c に変化すると，その n-gram 遷移確率は $P(p_i|t_{j-n+1}...t_{j-1})$ から $P(c|t_{j-n+1}...t_{j-1})$ に変化する．ここで，j は T の位置を表す．即ち， T の部分文字列用いて n-gram 遷移確率に基づく距離計算をする．n-gram 遷移確率情報は文書コーパスを元に算出する．コーパス中に出現する n-gram の頻度を基に確率値を求める．

置換

P 内の文字 p_i を T 内の文字 t_j に置換する．p_i までの n-gram の遷移確率は $P(p_i|t_{j-n+1}...t_{j-1})$ ，t_j までの n-gram の遷移確率は $P(t_j|t_{j-n+1}...t_{j-1})$ である．置換コスト δ_{rep} は以下の式で定義する．

$$\delta_{rep} = \frac{\ln P(t_j|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-n+1}...t_{j-1})} \tag{4.1}$$

$P(t_j|t_{j-n+1}...t_{j-1})$ の方が大きい場合，置換を行うことで「尤もらしい文字の並び」になったと考えられ，置換コストは低いものとなる．一方，$P(p_i|t_{j-n+1}...t_{j-1})$ の方が大きい場合,「生じにくい文字の並び」に近づくと考えられ，置換コストは高くなる．図 4.1 は“ $solt$ ”内で“ o ”から“ t ”へ遷移するところを置換操作により“ d ”への遷移に変更した場合を示す．3-gram 遷移確率を利用するとき，遷移文字変更により $P(t|ol)$ から $P(d|ol)$ に変化する．

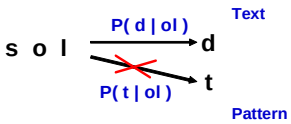

図 4.1: 遷移確率距離における置換操作

以下で述べる挿入・削除コストの式も同様に、操作前の確率が分子、操作後の確率が分母の形をしている．従って操作前の確率が低ければ操作コストは大きくなり，操作後の確率が高ければ操作コストは小さくなる．

挿入

P に生じる文字 p_i の前に T にある文字 t_j を挿入する．挿入前に続くときの文字 p_i までの n-gram の遷移確率は $P(p_i|t_{j-n+1}...t_{j-1})$ であり，挿入により後続文字として生じる文字 t_j までの n-gram の遷移確率は $P(t_j|t_{j-n+1}...t_{j-1})$ である．挿入コスト δ_{ins} は以下の式で与える．

$$\delta_{ins} = \frac{\ln P(t_j|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-n+1}...t_{j-1})} \tag{4.2}$$

図 4.2 は“ $plan$ ”で“ a ”と“ n ”の間に文字“ i ”を挿入し，遷移を変更する場合を示す．3-gram 遷移確率の場合，その確率は $P(n|la)$ から $P(i|la)$ に変化する．

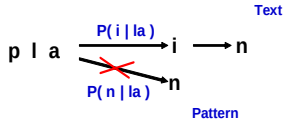

図 4.2: 遷移確率距離における挿入操作

対象となる文字と出現確率の形が変わらないため置換の式と同様の形となっている．

削除

P 内の文字 p_i を削除することで，後続文字は p_i から p_{i+1} に変化する．従って，p_i までの n-gram 遷移確率 $P(p_i|t_{j-n+1}...t_{j-1})$, p_{i+1} までの n-gram 遷移確率 $P(p_{i+1}|t_{j-n+1}...t_{j-1})$ を用いて，削除コスト δ_{del} を以下の式で定義する．

$$\delta_{del} = \frac{\ln P(p_{i+1}|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-n+1}...t_{j-1})} \tag{4.3}$$

図 4.3 は“ $plain$ ”で“ a ”から遷移する文字“ i ”を削除し，“ n ”に遷移を変更する場合を示す．

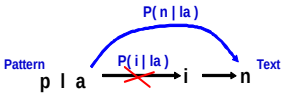

図 4.3: 遷移確率距離における削除操作

4.2.3 遷移確率距離を用いた DP 類似検索

DP により，パターン $P_{1...l}$ とテキスト $T_{1...m}$ における距離を遷移確率距離を用いて計算するアルゴリズムを示す [1]．列に P ，行に T を配置した行列 $C_{0...m,0...l}$ を作成し，その行列に以下の 3 つの式に基づいて値を埋めることにより P，T 間の遷移確率距離を求める．

$$C_{0,0} = 0 \tag{4.4}$$

$$C_{j,0} = C_{j-1,0} + \frac{\ln P(t_j|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-n+1}...t_{j-1})} \tag{4.5}$$

$$C_{0,i} = C_{0,i-1} + \frac{\ln P(p_{i+1}|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-n+1}...t_{j-1})} \tag{4.6}$$

$$C_{j,i} = min \begin{cases} C_{j-1,i-1} + \delta(p_i, t_j) \\ C_{j-1,i} + \frac{\ln P(t_j|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-n+1}...t_{j-1})} \\ C_{j,i-1} + \frac{\ln P(p_{i+1}|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-+1}...t_{j-1})} \end{cases} \tag{4.7}$$

(4.4) 式は初期値であり，距離 $C_{0,0}$ は 0 である．(4.5) 式は最上行の行列値に対する式であり，左に位置する行列値 $C_{j-1,0}$ に t_j を挿入する追加コストを示す．(4.6) 式は最左列にある行列値に対する式であり，上に位置する行列値 $C_{0,i-1}$ に p_i を削除する追加コストを示す．(4.7) 式は行列における左上・左・上の値があらかじめ計算されているとき，それらの値に操作コストを加えることによって $C_{j,i}$ が求まることを示す．$C_{j-1,i-1}$ から遷移した場合，比較文字列に文字 p_iと t_j が追加される．p_iと t_j が一致している場合，$\delta(p_i, t_j) = 0$ となる．一致していない場合，$\delta(p_i, t_j)$ は置換コストとなり，$\delta(p_i, t_j) = \frac{\ln P(t_j|t_{j-n+1}...t_{j-1})}{\ln P(p_i|t_{j-n+1}...t_{j-1})}$ となる．$C_{j-1,i}$ から遷移した場合は p_i が削除されていることを表し，$C_{j-1,i}$ に削除コストが追加される．$C_{j,i-1}$ から遷移した場合は t_j が挿入されていること表し，$C_{j,i-1}$ に挿入コストが追加される．これらの中で，コスト和が最小となる値を $C_{j,i}$ とする．

計算によって得られた行列値 $C_{j,i}$ は $P_{1...i}$ と $T_{1...j}$ の距離を表している．従って行・列が共に末尾である行列値 $C_{m,l}$ が P，T 間の距離を示す．

遷移確率距離において，各修正操作で扱う n-gram 遷移確率を 1 とすれば編集距離に一致する．第 2 章に示したパターン“ *abcd* ”，テキスト“ *zcde* ”の DP 計算の様子を図 4.4 に示す．この図は編集距離による DP 計算を示す。

この DP 計算法によるパターン類似度計算をテキスト集合に用いることで，類似検索の実現が可能になる．集合内の各テキスト T に対してそれぞれ DP 計算を行い，パターン P との距離を調べる．その中で最も距離の小さい，あるいは上位であるテキスト T を P と類似すると見なす．

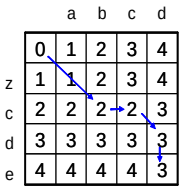

図 4.4: DP による編集距離計算

4.3 遷移確率距離の動的更新

時系列文書に対応するための遷移確率距離の動的更新手法について述べる．

前章で述べるように，遷移確率距離には n-gram 遷移確率を用いる．文書内で利用する文字を $c \in A$ （A はアルファベットを示す）とするとき，文字列 $C = \{c_1, c_2, \cdots, c_k\}$ 内のある n-gram の遷移確率 $P(c_i|c_{i-n+1}...c_{i-1})$ はその頻度 $f(c_i|c_{i-n+1}...c_{i-1})$ により，

$$P(c_i|c_{j-n+1}...c_{j-1}) = \frac{f(c_i|t_{i-n+1}...t_{i-1})}{\sum_{c \in A} f(c|c_{i-n+1}...t_{i-1})} \tag{4.8}$$

と計算できる．従って，n-gram 遷移確率はその頻度の更新に応じて動的に更新することが可能である．

時系列 $T = \{t_0, t_1, \cdots, t_\tau\}$ に対応する文書集合を $D = \{d_0, d_1, \cdots, d_\tau\}$ とするとき，各時刻の文書集合から得られる n-gram 頻度分布を $F = \{f_0, f_1, \cdots, f_\tau\}$ と示す．単純な更新法としては，時刻の変化と共に初期時刻 t_0 から時刻 $t_{\tau-1}$ の総和頻度分布に現在時刻 τ の頻度分布を加算する差分更新手法が考えられ，この差分更新頻度分布 f_{all} は，

$$f_{all}(c_i|c_{i-n+1}...t_{i-1}) = \sum_{t=0}^{\tau} f_t(c_i|c_{i-n+1}...t_{i-1}) \tag{4.9}$$

となる．しかし，t_0 から十分時間がたったある時刻 t では，総和頻度分布は分布としてほぼ収束しているため，現在時刻 τ から得た頻度分布 f_τ による差分更新の影響は限りなく小さく，現在の局所的な確率変動を捉えるのは困難である．例えば，時刻 τ で“ 大統領の辞任 ”というニュース記事が増加しても，それに関する n-gram は今までの総和頻度分布の n-gram に比べれば小さいものであり，距離に反映することができない．

一方，現在時刻 τ の頻度分布 f_τ を用いた場合，もし全体としての文の長さが短ければ十分な n-gram 頻度分布が得られず，検索精度は上がらない．

それらの問題点による検索精度への影響を抑制するため，本手法では差分更新頻度分布 f_{all} による遷移確率 P_{all} と現在時刻 τ の頻度分布 f_τn による遷移確率 P_τ を混合して遷移確率距離に用いる．この遷移確率を混合遷移確率 P_{mix} とし，

$$P_{mix} = \lambda P_\tau + (1 - \lambda) P_{all} \tag{4.10}$$

とする．この式における λ は現在定数をと示し，λ が大きいほど現在時刻における遷移確率を反映する混合遷移確率となる．

4.4 実験

4.4.1 実験準備

初めに，遷移確率距離計算に用いる 3-gram 遷移確率情報をロイターコーパス (Reuters Corpus) から得る．このコーパスは XML 形式のロイター新聞記事 1 年分（1996 年 8 月 20 日〜 1997 年 8 月 20 日）のデータである．“ スポーツ ”,“ 政治 ”トピック記事の本文にあたる text タグ内のデータのみ抽出し，すべて小文字に変換し，不要語を排除してから各々のトピックごとで 3-gram 遷移確率を計算する．不要語の排除は，どの文書でも多用する語であることから各時刻での遷移確率の差異を求める妨げになるという理由からである．ロイターコーパス内には 56 個の文字・記号が存在した（表 4.1）．

表 4.1: ロイターコーパス内に存在する文字

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7
8	9	0		!	”	$	’	(	)	*	+	,	-	.	/	:
;	=	?	^	~												

また，検索対象となるテキスト集合として，本実験では英辞郎辞書コーパスの見出し語を用いる．ロイターコーパス内の 56 個以外の文字・記号に対しては遷移確率が存在せず，距離計算が行えないため，それらの文字が含まれる単語は除外する．この結果，英辞郎辞書コーパスから抽出する単語数は 294,642 個となる．

4.4.2 実験方法

検索精度の評価実験を行う．エラー文字列をパターン P として与え，P と最小距離である単語，つまり最も類似する単語を英単語集合から検索する．その検索によりエラー前の正しい単語を類似解として得られるかどうかを，編集距離による類似検索，予め全日付の文書から確率を得た非更新型遷移確率距離による類似検索，提案手法である更新型遷移確率距離による類似検索でそれぞれ調べ，比較する．更新型遷移確率距離では現在定数 $\lambda = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$ の場合でそれぞれ実験を行う．エラー文字列にはロイターコーパスの各日付で得られた頻出単語上位 10 個に編集距離 1 でエラーを生成したものを用いる．各日付ごとに得られた検索結果から適合率 $Precision$・

再現率 $Recall$・F 値を以下の式に基づいて求める．

$$Precision = \frac{A}{C}, Recall = \frac{A}{B}, F = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{4.11}$$

式中での A, B, C はそれぞれ正解単語数，全正解数（常に 10），検索により類似と見なした単語数を表す．編集距離においては同じ距離の単語が複数検索解に挙げられることが考えられるが，その場合はすべて類似とみなす．従って C の値が大きくなり，適合率は低下する．

4.4.3 実験結果

各類似検索手法においてスポーツ・政治トピックにおける日付に対する平均適合率・再現率・F 値を表 4.2，4.3 に示す．

表 4.2: スポーツトピックにおける日付に対する平均の適合率・再現率・F 値

検索手法	平均適合率 [%]	平均再現率 [%]	平均 F 値 [%]
編集距離	20.46	**96.04**	30.67
非更新遷移	45.63	48.74	47.08
更新遷移（$\lambda = 0.0$）	45.25	48.16	46.61
更新遷移（$\lambda = 0.2$）	47.50	50.47	48.88
更新遷移（$\lambda = 0.4$）	48.03	51.10	49.46
更新遷移（$\lambda = 0.6$）	48.65	51.76	50.10
更新遷移（$\lambda = 0.8$）	**49.08**	**52.25**	**50.55**
更新遷移（$\lambda = 1.0$）	46.65	49.64	48.04

表 4.3: 政治トピックにおける日付に対する平均の適合率・再現率・F 値

検索手法	平均適合率 [%]	平均再現率 [%]	平均 F 値 [%]
編集距離	39.05	**97.10**	53.77
非更新遷移	66.78	69.01	67.83
更新遷移（$\lambda = 0.0$）	66.97	66.97	68.04
更新遷移（$\lambda = 0.2$）	65.32	66.96	66.09
更新遷移（$\lambda = 0.4$）	65.59	67.32	66.40
更新遷移（$\lambda = 0.6$）	69.41	71.70	70.49
更新遷移（$\lambda = 0.8$）	**69.54**	**71.84**	**70.62**
更新遷移（$\lambda = 1.0$）	66.15	67.97	67.01

適合率・F 値において，編集距離より遷移確率距離を用いた場合の方が高い値を得た．編集距離で再現率が高いのは元々編集距離 1 でエラーを与えたという設定からで

ある．また非更新型に比べて更新型の遷移確率距離では $\lambda = 0.8$ で検索性能の改善が見られ適合率・再現率・F 値においてすべて約 4% 程度高い値となっている．

4.4.4 考察

更新型遷移確率距離において，現在時刻のみの遷移確率である $\lambda = 1.0$ のときに比べて $\lambda = 0.8$ の場合の方が精度が改善した理由について考える．表 4.4 は，スポーツトピックにおいて 10 日ごとの適合率の推移を示したものである．パターンが 10 個のみなので λ の変化によっての差は極端ではない．しかし，差があるところを見ると，$\lambda = 1.0$ の場合のみ適合率が低い日が時々現れている．Over-fitting による影響だと考えられる．また，$\lambda = 1.0$ が最も高い適合率の場合は $\lambda = 0.8$ も同様に最も高い適合率を示していることが多い．従って平均をとった結果，$\lambda = 0.8$ のときが最も良い適合率になったと考えられる．

日に関係なく常に $\lambda = 0.8$ の場合で最も精度が良いというわけではなかった．時刻に応じて得られる文書の大きさは異なる．それに対応して，最も良い精度を示す λ の値が大小に変化すると思われる．

4.5 結論

本研究では文字列距離の動的更新により時系列文書に適応する類似検索手法を実現した．文書内容の変化に伴う文字列中の文字遷移確率の変化に着目し，著者らの提案した遷移確率距離 [8] に適用することでより時系列文書に対しての検索精度を 約 4% 改善した．従来提案されている文字列距離では動的更新を考慮するものは少なく，Web 上から膨大な情報が得られる現代において，他の文字列距離よりも有用であると考えられる．

本研究では差分更新した総時刻頻度分布と現在時刻頻度分布の今後に対して現在定数を用いて混合を行ったが，その時刻ごとで得られる文書の大きさは異なるはずであり，それに対応してその値を変動させることでより精度を改善できる可能性がある．今後さらなる実験を行うことで，動的更新手法の高度化を行っていきたい．

表 4.4: 適合率の推移

	更新遷移 [λ]					
日付	0.0	0.2	0.4	0.6	0.8	1.0
1996/8/20	54.55	54.55	54.55	54.55	54.55	45.45
1996/8/30	27.27	36.36	36.36	36.36	36.36	36.36
1996/9/9	33.33	33.33	36.36	36.36	36.36	27.27
1996/9/19	40.00	50.00	50.00	50.00	50.00	50.00
1996/9/29	33.33	33.33	33.33	33.33	33.33	33.33
1996/10/9	45.45	45.45	45.45	45.45	45.45	45.45
1996/10/19	60.00	60.00	60.00	60.00	60.00	40.00
1996/10/29	27.27	36.36	36.36	36.36	36.36	36.36
1996/11/8	50.00	50.00	50.00	50.00	50.00	40.00
1996/11/18	50.00	60.00	60.00	60.00	60.00	60.00
1996/11/28	45.45	45.45	54.55	54.55	54.55	54.55
1996/12/8	50.00	50.00	50.00	50.00	50.00	50.00
1996/12/18	72.73	72.73	63.63	63.63	63.63	63.63
1996/12/28	33.33	33.33	33.33	33.33	41.67	41.67
1996/1/7	45.45	45.45	45.45	45.45	45.45	45.45
1996/1/17	25.00	33.33	33.33	33.33	33.33	33.33
1996/1/27	45.45	45.45	45.45	36.36	36.36	27.27
1996/2/6	70.00	70.00	70.00	70.00	70.00	70.00
1996/2/16	50.00	50.00	50.00	50.00	50.00	40.00
1996/2/26	41.67	50.00	50.00	50.00	50.00	50.00
1996/3/8	36.36	36.36	36.36	36.36	36.36	36.36
1996/3/18	40.00	40.00	40.00	40.00	40.00	40.00
1996/3/28	45.45	36.36	36.36	27.27	27.27	27.27
1996/4/7	50.00	50.00	50.00	50.00	50.00	50.00
1996/4/17	50.00	50.00	40.00	40.00	40.00	40.00
1996/4/27	40.00	40.00	40.00	40.00	40.00	40.00
1996/5/7	50.00	63.63	63.63	63.63	63.63	63.63
1996/5/17	45.45	45.45	45.45	45.45	45.45	45.45
1996/5/27	40.00	40.00	40.00	40.00	40.00	40.00
1996/6/6	36.36	40.00	50.00	50.00	50.00	40.00
1996/6/16	20.00	20.00	20.00	20.00	20.00	10.00
1996/6/26	50.00	50.00	50.00	50.00	50.00	50.00
1996/7/6	54.55	54.55	54.55	54.55	54.55	54.55
1996/7/16	50.00	60.00	60.00	60.00	70.00	70.00
1996/7/26	58.33	58.33	58.33	58.33	58.33	50.00
1996/8/5	50.00	60.00	60.00	60.00	70.00	70.00
1996/8/15	30.00	30.00	30.00	30.00	36.36	36.36

第5章　結論

本研究では，まず多次元文字データに対する文字列類似検索の有用性を示し，その結果からより有用性を高めるために，文字遷移確率を用いた類似検索手法を提案し，検索性能の向上を実現した．

まず，多次元文字データにおける類似検索では，文字データを 3 次元空間上で扱える 3 次元エディタにより，本来 1 次元データである文字データの多次元へのモデル化を実現した．このデータから断面として 2 次元文字データを抽出し，縦横への類似文字列検索により，1 次元データでは得られなかった類似検索解の獲得を本実験により示した．選択する断面により 2 次元データが意味する情報が異なるため，そこから得られた類似検索解が表現する情報も変化する．このことから，多次元文字データとして文字データを扱うことによって，1 次元データよりも多様な形で類似検索解が得られるということを実証できた．

次に，文字遷移確率を用いた類似検索では，文字列中の文字の遷移確率を文字列距離に適用することにより，文字データにおけるマルコフ過程的な遷移特徴に基づいて類似検索解を得る手法を提案した．編集距離等を用いた類似検索において問題であった同一距離の類似解の発生を極端に抑えることができ，容易に検索ユーザーの最も意図する文字列を推定することが可能になった．実際に編集距離を用いた類似検索手法と比較して，精度は約 30〜38% 向上した．

遷移確率を動的更新させた類似検索では，時系列文書に適応した遷移確率距離を提案し，その時刻ごとに最も意図する類似検索解を自動推定する手法を実現した．従来提案された文字列距離には取り入れられていなかったが，距離の動的更新という概念を加えることで，より強力な距離関数となり，情報が日々更新されるこの情報化社会において有用なものであることを実験により示すことができた．この実験では，編集距離での類似検索手法に対して約 31%，非更新遷移確率距離での類似検索手法に対して約 4% の精度の改善が見られた．

本研究では，遷移確率による類似検索において，一般的に利用される新聞記事等の文書データを扱って精度の向上が実現できたかどうかを示した．最初に提案した多次元文字データに対しても応用できるので，検索精度の向上ができるはずである．また，遷移確率距離の手法は，シンプルな形で提案した上で有用な結果を得ている．高度な手法を取り入れることで，より高精度な類似解の自動推定が可能になると考えられる．

謝辞

本研究を遂行するにあたり，日頃より適切な御指導，御鞭撻をいただいた，法政大学工学部情報電気電子工学科 三浦孝夫教授に深く御礼申し上げます.

並びに，産能大学経営情報学科 塩谷勇教授にも多くの有益なご助言をいただきました．深く感謝いたします.

データ工学研究室の先輩方，同輩，後輩たちにも，本研究の遂行にあたって数多くの助言と快適で能率的な研究室環境を整えていただきました．御礼申し上げます.

修士論文として私の研究をまとめることができたのも，多くの皆様方の御支援，御協力の賜物であります．この場をお借りしまして，厚く御礼申し上げます.

最後に，今までの学生生活を支えてくださった私の両親・兄弟に深く感謝したいと思います.

参考文献

[1] Navarro, G.: "A Guided Tour to Approximate String Matching", ACM Comp. Surveys, 33(1), pp. 31-88, 2001

[2] Levenshtein, V.I.:"Binary codes capable of correcting spurious insertions and deletions of ones", Problems of Information Transmission, 1(1), pp. 8-17, 1965

[3] 北 研二, 津田和彦, 獅々堀正幹:"情報検索アルゴリズム", 共立出版, 2002

[4] Cohen, J.: "Bioinformatics—an introduction for computer scientists", ACM Comp. Surveys, 36(2), pp. 122-158, 2004

[5] Altschul, S.F., Gish, W., Miller, W., Myers, E.W.& Lipman, D.J.: "Basic local alignment search tool", J.Mol.Biol. 215(3), pp. 403-410, 1990

[6] Wei, J.: "Markov Edit Distance", IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(3), pp. 311-321, 2004

[7] Ristad, E.S., Yianilos, P.N.: "Learning String Edit Distance", IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(5), pp. 522-532, 1998

[8] Katsumata, A., Miura, T., Shioya, I.:"Approximate String Matching Using Markovian Distance", Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), 2010

[9] "The story behind Eijiro: the most popular Japanese/English dictionary on the net", www.stippy.com/japan-language/the-story-behind-eijiro-the-most-popular-japanese-english-dictionary-on-the-net/