解 説

類似検索アルゴリズムを上手に利用する

Good Utilities of Similarity Search Algorithms

宇 野 毅 明* *国立情報学研究所

Takeaki Uno* *National Institute of Informatics

検索アルゴリズムは、情報処理の中でも特に基礎的かつ汎用性の高いタスクであり、広く一般に使われている。ロボットを含む機械・制御や人工知能といった分野でも多く使われており、だれもが一度は利用する手法であろう。

一般に検索というと、ある「もの」をデータベースの中から「探し出す」とことと見なされることが多い。WebサービスなどのIT分野でも、例えばユーザの検索、キーワード検索などがこれに相当する。ロボットでは、例えば、撮影した画像の中の物体がデータベースに存在するかを調べる、といったように、観測して得られたデータが既知のどの情報と一致するかを調べる。これは、化学情報や生物情報などの分野でも同様である。ざっくりばらんに言えば、検索を用いるアプリケーションの8割以上がこの目的であろう。

一方で、残りの2割の中には、データベースという背景知識を用いないアプリケーションも存在する。例えば、観測したデータの物体を、似たもの同士でクラスタリングしたい場合（教師なしクラスタリング）がこれに相当する。観測データから物体の辞書を作るタスクはこのようなクラスタリングを通じて行われ、他の分野、画像処理や自然言語処理でも同様である。この場合、検索はデータベースに対して行われるのではなく、獲得した多量のデータに対して、データ同士で類似するものを見つけることになる。つまり、データが $a_1, \dots, a_n$ で構成されているときに、すべてのペア a_i と a_j に対して同一性、あるいは類似性を計算する。このほかにも、複数の画像中の物体の対応付けをしたい、動きのパターンを類別したい、過去の行動ログから多数現れる異常を発見したい、といった目的のために使われている。

このようなアプリケーションでは、一般にキーワード検索で行われるような「完全な一致」を求めるだけでは十分でないことが多い。一般に、違う角度で、異なる時刻に撮影した画像は、たとえ同じ物体を撮影したものであっても異なってしまう、その画像特徴量もある程度異なってくる。他のセンサデータや行動ログでも、同じ意味を持つ物がまっ

たく同じ表現がなされることは少なく、必ずといっていいほど「揺らぎ」が内包されている。このような状況では、完全一致は不十分であり、類似検索が用いられる。「まったく同じ物」ではなく「一番似ている物」を見つけるのである。このように揺らぎを解消するのが、最近のデータ解析手法の潮流である。このような距離の利用は非常に多くの対象、音声、画像、位置、時系列、形状、ログなどについて用いられており、情報を用いたデータ解析でも大きな位置を占めている。

学習をはじめとする解析アルゴリズムでは、アイテム（データ点）間の距離を計算して距離行列を計算し、それをもとに計算を行うものが多い。このとき、完全な距離行列、つまり任意のデータ点間の距離を求めるようにすると、巨大なデータでは時間的・記憶容量的にコストが巨大になる。そのため、一般には短い距離のみが記録された、疎な行列を用いることが多い。この行列を求める問題を「疎な距離行列を求める問題」と呼ぼう。

一般に、距離を用いるデータ解析では、長い距離というのは情報を持たないことが多い。データ点が球面上に分布している場合などは、ある点から最も遠い点が一意に定義され、その周辺はある種の意味を持つと考えられる。しかし、ユークリッド空間に分布するデータ点などでは、与えられたデータ点から遠くにある要素は一箇所に集中せずばらばらに分布しており、それらに何らかの共通な意味を見つけることは難しい。こういった意味から、距離を用いたモデルを考える場合「短い距離」は意味を持つことが多く、その値にもなんらかの強弱を認めることができるが、長い距離は単に「距離が短くない」という意味しか持たないことが多い。そのため、多くの解析アルゴリズムは短い距離しか利用しないように設計が可能であり、大規模なデータの取り扱いが可能になるとともに、小さい問題でも計算時間を格段に短くすることが可能となるのである。

このような距離行列を計算する、あるいは単に短い距離のデータ点の対を見つけたい、という問題は、一般には類似検索であるとみなされる。しかしこの類似検索が一筋縄ではいかない問題であり、情報技術の中でも効率化が難しい問題の一つである。一般にキーワード検索などの「完全

原稿受付 2014年12月1日

キーワード: Similarity, Algorithm, Index, Search, Multi-sorting Algorithm

〒101-8430 東京都千代田区一ツ橋 2-1-2

*Chiyoda-ku, Tokyo

一致」を求める検索問題に対しては、二分探索やハッシュなど、非常に効率的な技術が数多く存在する。これはひとえに、検索する対象に「推移律が成り立つ」と「全順序がある」ことが影響している。完全一致では推移律が成り立つため、「AとBが同じであれば、両者は同じ特徴（例えばハッシュ値）を持つ」という大変利便性の高い性質が成り立つ。そのため、ハッシュ値を分類のキーとして用いることで、効率的な検索ができるのである。さらに全順序がある場合は、辞書を引くときに用いる二分探索をすることで、質問点に対して最も近い点をデータの中から探し出すことも、効率良くできる。しかし、類似関係では推移律が成り立たないため、「AとBが類似していても、両者の特徴は同じであるとは限らない」ので難しい。例えば英単語の特徴をその先頭の文字だと思えば、完全一致では、同じ単語は必ず先頭の文字は同じだが、似た英単語でも、先頭の文字が異なる場合はいくらでもある。類似性に関して全順序が成り立たない場合、例えば二次元平面にデータ点が分布している場合、平面上で距離が近い、という関係は全順序ではない。距離が近い物が全順序の関係で必ず近くなるような順序づけをすることは不可能である。よって、二分探索などの技術を直接的に利用することができない。図1は、全順序が成り立つ場合に木型の索引をデータ上に付与した様子の模式図である。下の平たい四角がデータであり、全順序の順番にソートされている。その上に二分探索の動きを構造化した木構造の探索路が構築されており、この木を図のように上から下へとたどることで二分探索と同じ効率でデータ点を探し出すことができる上、全順序で近い位置にあるデータ点も短時間で見つけることができる。しかし、全順序がない場合、どのように順序づけしても、与えられた点に近いデータ点は図の黒丸のように分散してしまい、効率良く見つけることができない。

これらの理由から、一般に類似検索には様々な工夫を凝らしたアルゴリズムが用いられている。しかし、なんにしても単純な解決法は存在しないため、検索の効率を上げるためには計算コストとメモリコストの高い索引作りが必要となっており、これが計算上のボトルネックになっている。精度を保証するためには計算時間も長大になり、一般には精度保証のない、ある種のヒューリスティクスを用いた、確

図1 二分探索を模した索引構造とその辿り方。下の黒丸は、ある点に近いデータ点の例

率的なアルゴリズムが用いられている。これが、現実問題での類似検索の利用に対する大きな障害となっている。

このように、単一の類似検索でもコストが大きいことから、疎な距離行列を求めるには莫大なコストがかかりそうに思える。しかし、実際にはコストを減らす方法がたくさんある。例として、正確な一致を見つける場合に対する高速化手法を見てみよう。まず、データに含まれるデータ点を $a_1, \dots, a_n$ をなんらかの全順序でソートする。すると、同じ値を持つデータは必ずソート列の中に連続して現れるため、すべての a_i に対して簡単に同じ値を持つものを見つけることができる。つまり、通常の検索では索引の構築と索引をたどる手間がかかるのに対して、この方法ではそれらが単にソートをするだけになっている。通常、ソートは他の処理よりも簡便にかつ高速で行えるため、非常に効率が高くなる。注意したいのは、この方法は一つのデータ点に対する検索を行う目的には向かないということである。データ全体をソートするため、質問が一つであるにもかかわらず非常に長い時間を要してしまう。この方法が活きるのは疎な距離行列の構築など質問データ点が多い場合であり、まとめて計算を行うスケールメリットがある場合に限られる。この種の工夫は、類似検索にも応用できるため、索引を用いた類似検索技術の利用よりも効率的に疎な距離行列の構築ができるのである。

ここから先は、筆者が開発した、複数ソート法[1]の解説を行おう。複数ソート法は、文字列の類似性に対する手法であり、類似検索手法の中でも最初にこのような考え方をういたアルゴリズムである。入力として、長さ l である文字列が n 本与えられたとしよう。そして、ハミング距離を類似性として閾値 d を与え、ハミング距離が d 以下であるような a_i, a_j の組をすべて求める問題を考える。これは、文字列データに関する疎な距離行列の、小さい距離を持つ要素の場所を列挙していることに等しい。

データを文字列に限定することは非常に問題の範囲を狭めているように感じるかも知れないが、実際はそうではない。例えばLSH（ローカルセンシティブハッシュ）という技術があるが、これはユークリッド空間などの上分布するデータ点に対してビット列を与える方法であり、距離が近い点は似たビット列が割り当てられるようになっている。また、SASHという類似検索手法では、ランダムに選んだサンプル点を用いてデータ点を特徴付けており（一番近いサンプル点の特徴となる）、似たようなアイデアでビット列が構築できる。これらのビット列は、ビット列が似ていればデータ点の距離は近くなることが多い、という性質を満たすため、文字列の類似性を調べることで、もとのデータ点の類似性を調べることができる。

アルゴリズムのアイデアは、以下の性質を利用することである。 S と T を長さ l の文字列とし、 $S_1, \dots, S_k$ と

図2 二分 $d = 2$, $k = 5$ の場合の例; 異なりがある1番めと4番めのブロック以外は等しい

$T_1, \dots, T_k$ を, S_i と T_i を等しい長さの k 個 ($k > d$ とする) のブロックに分割したものとする. このとき, S と T のハミング距離が d 以下であれば, S と T の少なくとも $k - d$ 個のブロックが等しくなる. ハミング距離が d 以下であれば, 異なる文字が d 箇所しかないため, たかだか d 個のブロックしか異なることができない, ということから明らかである (図2 参照). この性質から, ハミング距離が d 以下の文字列ペアを見つけるには, $k - d$ 個のブロックが等しい文字列ペアだけを調べれば良いことが分かる.

次に, $k - d$ 個の等しいブロックを持つ文字列ペアを見つける作業を高速に行う方法を解説しよう. キーとなるアイデアは, $k - d$ 個の等しいブロックの位置を何番めと何番めであるのか, という形でまず指定してしまい, その位置に等しいブロックを持つペアを見つける, というものである. 図2の例で言えば, 等しいブロックの位置は2, 4, 5番めである. ほかに, 1, 2, 3番めが等しい場合もあるし, 1, 2, 3, 4, 5番めのすべてのブロックが等しい場合もあるだろう. これらを場合分けして, 1, 2, 3番めのブロックが等しいペアを見つける, 1, 2, 4番めのブロックが等しいペアを見つける, 1, 2, 5番めの, というようにすべての組み合わせを尽くし, 3個のブロックが等しいペアをすべて見つけるのである.

いったん等しいブロックの位置を決めてしまうと, $k - d$ 個のブロックが等しい文字列ペアは, 簡単に見つけることができる. 例えば, それらの位置にあるブロックをハッシュキーとしてハッシュを構築しても良いが, ここではさらに効率の良い, ソートを使った方法を紹介する. データ中のすべての文字列を, 与えられた $k - d$ 個のブロックが作る部分文字列をキーにしてソートしよう. すると, $k - d$ 個のブロックが同じものは, ソートされた列の中でかたまりとなって存在するようになる. つまり, ソートすることによって, データのすべての文字列は, 「 $k - d$ 個のブロックが同じ文字列もの」のグループに分類されるのである (上記の正確な一致の場合の例と同じである). あとは, 各グループについて全対比較を行って, ハミング距離が d 以下のペアを探せば良い.

このアルゴリズムの効率は, k の選び方に大きく依存する. ソートの回数は dC_k 回である. 各グループの大きさは,

図3 1, 3, 4番めのブロックをキーにして文字列をソートした様子; 1, 3, 4番めのブロックが同じである文字列がかたまる

文字の分布にある程度の一様性を仮定すれば, 比較する文字列の数を「文字の種類数の $l(k - d)/k$ 乗」で割ったものになる. l がそれなりに大きく d がそれなりに小さければ, おおざっぱな見積もりでちょうど良い k を選ぶことで, 計算時間はほぼ線形時間になる. 計算時間の見積もりは, 現実のデータでもそれほど大きく外すことはないので, 直接的に計算した場合の2乗時間と比べるとはるかに高速化される.

実際にこのアルゴリズムを使って, ゲノムの解析を行った例を以下に示そう. ゲノムは非常に長い文字列であるが, 30 程度の長さの部分文字列をすべて取り出すことにより, 同じ長さの文字列の比較によって, 部分的な類似性を検出できる. つまり, 類似する文字列AとBがあるとき, Aの長さ30の部分文字列とBの長さ30の部分文字列でハミング距離が短いペアが沢山あるはずである, という現実的仮定に基づいたモデルである. このようなゲノムの類似性の検索は, バイオ情報学における中心的な情報処理であり, ほぼすべての解析に何らかの形でかかわっていると言ってもいいだろう.

ゲノムの類似検索に対しては, BLAST というアルゴリズムが有名であり, このアルゴリズムの用いているモデルが業界の事実上の標準規格になっている. BLASTのアイデアも上述のアイデアと似ているのであるが, ハミング距離の短いものを見つけるのではなく, 正確に一致するような部分文字列の組を見つけているところが異なる. これは小さな違いのように思えるかも知れないが, 実際には非常に大きな効率の違いを生んでいる. 例えば, 1割程度の誤差を許した類似検索をしたい, つまりハミング距離が文字列の長さの $1/10$ くらいであるようなものを見つけないとしよう. そうすると, 文字列AとBがこの条件, ハミング距離が長さの1割, を満たしたとしても, 長さ10以上の正確に一致する部分文字列は含まないかも知れない. 当然, A, Bがある程度長ければ, 少々長いものも高い確率で含むであろうが, それでもそれほど長いものは期待でき

図4 マウスのX染色体（横）と人のX染色体（縦）の比較；似ている部分文字列が多い部分に対応するセルが色づいている

ない。そうすると、長さ10程度の正確な一致をゲノムから見つけることになるが、ここで一つ注意しなければいけないのは、ゲノムはATGCという4種類の文字のみから構成されているということである。つまり、文字種が少ないため、例えば1億本の文字列を比較した場合、 $1\text{億}/100\text{万}=100$ （100万は4の10乗の概数）程度は同一な物があるということである。実際のデータはより分布がゆがんでいるため、1万程度と見積もるべきだろう。そうすると、1億本に対して、1万本の比較を行うことになり、これはとても短時間で済ませられるものではない。

一方で、我々の方法であれば、例えば長さ30でハミング距離が2、といったものを見つけることになる（これで、ハミング距離1割の条件は満たす）ここで分割数を5に

すれば、3ブロックが等しいものを見つけることになり、 $30 \times (3/5) = 18$ 文字の一致で比較対象を絞り込むことになる。これにより、比較対象はおよそ $4^8 = 65536$ 分の1程度になると見積もられるため、非常に計算が高速になる（実際はもう少し悪いだろう）。実際に、マウスのX染色体と人のX染色体を比較したときは、2～3週間かかる計算をわずか15分程度で終了させることができた。比較結果の絵を図4に示す。

本稿では、類似検索の有用性と難しさと、拝啓知識を用いないタイプの検索では一気にすべてを比較することにより簡単な高速化が行えることを紹介した。合わせて、複数ソート法による多量の文字列の全対比較を高速に行う方法を紹介した。情報的な問題には、モデルとアルゴリズムの工夫により飛躍的な性能向上が行えるものが数多くある。情報技術の効果的な利用には、このようなモデルとアルゴリズムの視点が非常に重要であり、読者のそのような感覚の醸成に本稿が資することができれば幸いである。

参考文献

- [1] T. Uno: "Multi-sorting algorithm for finding pairs of similar short substrings from large-scale string data," Knowledge and Information Systems, vol.25, no.2, pp.229–251, 2010.

宇野毅明 (Takeaki Uno)

1998年東京工業大学システム科学専攻博士（理学）を取得。同年同経営工学専攻助手着任、2001年国立情報学研究所助教授着任。現在は教授。専門はアルゴリズム理論とその応用、特に離散・列挙・組合せ最適化、データマイニング・データ解析での大規模高速計算。2010年文部科学大臣表彰若手科学者賞受賞。