1M-02

整数線形計画法による文字列の集合上の確率分布における中央文字列探索の高速化

槇原　このか　　林田　守広　　小谷野　仁
松江工業高等専門学校専攻科　　松江工業高等専門学校　　農業・食品産業技術総合研究機構
電子情報システム工学専攻　　電気情報工学科

１．はじめに

文字列の集合の場合、中央文字列がデータの中心として用いられる。データの中心とは、数データの場合は平均であり、データの特徴を調べる最も基本的な尺度である。中央文字列は、集合に含まれる各文字列との距離の和を最小にする文字列である[1)]。中央文字列は、生物進化においては共通祖先の配列を見つける目的に利用でき、またたんぱく質配列に対しては機能モチーフの同定に利用できる。さらに生物学的配列の分類やクラスタリングにも適用例がある。

Hayashida ら[2)]は、レーベンシュタイン距離を計算するアルゴリズムに基づいて、整数線形計画問題を定式化した。本研究では、中央文字列の計算時間短縮を目的として、整数線形計画問題の改良を試みる。

２．提案手法

2.1　中央文字列

$A=\{a_1 \cdots a_z\}$をzの異なる文字の集合とし、A^*をAの文字を任意有限個並べた文字列の集合とする。集合A^*上の確率分布$p(s)$が与えられたとき、$p(s)$に対する中央文字列は式(1)を最小化する$t(\in A^*)$と定義される。$d(t,s)$は文字列sとtの距離を示す。

$$\sum_{s\in A*} p(s)d(t,s) \qquad (1)$$

2.2　レーベンシュタイン距離

レーベンシュタイン距離は、ある文字列sからtまでの編集の最小コストで定義する。編集操作には置換、削除、挿入の編集操作があり、本研究では、各操作コストを 1 とする。図 1 に、krelo と hello のレーベンシュタイン距離の求め方の例を示す。一致していない文字 k-h を置換する。不要な文字 r を削除する。不足している文字 l を挿入する。これより操作は減らせないため、krelo と hello の文字列の距離は 3 となる。

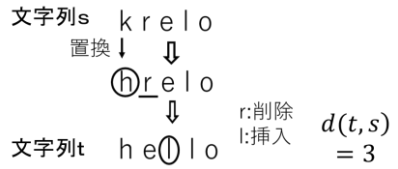

図 1　レーベンシュタイン距離の求め方例

2.3　整数線形計画問題の定式化

一致する文字数、または置換して一致する文字数が最大となる中央文字列を求める問題を、式(2)に示す整数線形計画問題として定式化した。N 個の文字列$s^{(k)}(k=1,\cdots,N)$に対する確率$p(s^{(k)})$が与えられたとき、中央文字列$t=t_1\cdots t_n$が決定される。中央文字列候補の文字数は、$\sum_{k=1}^{N}|s^{(k)}|$以下で十分である。$|s^{(k)}|$は文字列$s^{(k)}$の文字数を示す。ここで、$m_{ij}^{(k)}$は$s_i^{(k)}$とt_jが一致すれば 1、そうでなければ 0 となる変数を示す。

$$\left.\begin{aligned}
&\text{最大化}\\
&\sum_{k=1}^{N} p(s^{(k)}) \sum_{(i,j)\in I^{(k)}\times J} (m_{ij}^{(k)}+h_{ij}^{(k)}) - n\\
&\text{制約式}\\
&\sum_{j\in J} m_{ij}^{(k)} \le 1, \qquad \sum_{i\in I^{(k)}} m_{ij}^{(k)} \le 1,\\
&m_{ij}^{(k)} + m_{i'j'}^{(k)} \le 1\big((i<i'\wedge j'<j)\vee(i'<i\wedge j<j')\big),\\
&h_{ij}^{(k)} \le m_{ij}^{(k)},\ h_{ij}^{(k)} + g_{ij}^{(k)} \le 1,\ s_i^{(k)} - t_j \le |A| g_{ij}^{(k)},\\
&t_j - s_i^{(k)} \le |A| g_{ij}^{(k)}, \qquad j\times m_{ij}^{(k)} \le n,\\
&j\times m_{1j}^{(k)} \le b^{(k)}, \qquad \frac{n}{2} < b^{(k)}, \qquad \frac{2n-|s^{(k)}|}{2} < b^{(k)},\\
&i\times m_{i1}^{(k)} \le c^{(k)}, \qquad \frac{|s^{(k)}|}{2} < c^{(k)}, \qquad \frac{2|s^{(k)}|-n}{2} < c^{(k)},\\
&m_{ij}^{(k)}, h_{ij}^{(k)}, g_{ij}^{(k)} \in \{0,1\}, \qquad t_j \in \{1,\cdots,|A|\}\\
&I^{(k)} = \{1,\cdots,|s^{(k)}|\}, \qquad J=\{1,\cdots,n\}
\end{aligned}\right\} \qquad (2)$$

$s^{(k)}$とtのレーベンシュタイン距離の上限はす

Acceleration of Finding Median Strings in Probability Distribution on a Set of Strings by Integer Linear Programming
Konoka Makihara, National Institute of Technology, Matsue College
Morihiro Hayashida, National Institute of Technology, Matsue College
Koyano Hitoshi, The National Agriculture and Food Research Organization

べての文字を置換操作した場合であるので

$$d(s^{(k)},t) \le \max\{|s^{(k)}|,|t|\} \quad (3)$$

が成り立つ。中央文字列の解候補を削減すると、計算時間短縮につながる。そこで、文字列がある程度長い図 2 の文字列を考える。$s^{(k)}$の先端の文字とtの$(b+1)$文字目がマッチする場合を考えると、a文字の削除、b文字の挿入操作を要する。従って、距離は$a = |s^{(k)}| - (|t| - b)$より

$$\begin{aligned} d(s^{(k)},t) &\ge a + b \\ &\ge |s^{(k)}| - |t| + 2b \end{aligned} \quad (4)$$

となる。レーベンシュタイン距離は編集操作の最小コストであるから、式(4)の距離が式(3)の上限より大きくならないため、解候補から除くべきである。このような$m_{ij}^{(k)}$=1(j>b)のbを式(5)により決定する。

$$\max\{|s^{(k)}|,|t|\} < |s^{(k)}| - |t| + 2b \quad (5)$$

$(b+1)$以上の文字と 1 文字目がマッチする解候補は、削除するよう式(2)の 6 行目に定めた。中央文字列の 1 文字目も同様な条件で一致しないよう、制約を式(2)の 7 行目に定めている。

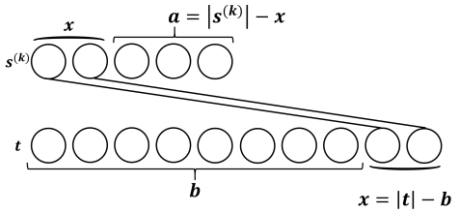

図 2 入力文字列$\boldsymbol{s^{(k)}}$と中央文字列$\boldsymbol{t}$のマッチング

3．実験方法

入力文字列 2 本、文字数 3～4 文字の条件について、全数探索プログラムを用いて、出力された中央文字列が正しいことを確認した。

計算機実験を行い、計算時間を評価した。文字の集合の要素数$|A|$=4 とした。$p(s)$>0 を満たすN本の長さ$|s^{(k)}|$の文字列$s^{(k)}$をランダムに生成した。ここでN=2,…,10、n_k=2,…,6 とした。$s_i^{(k)}$は、αを平均 0、分散 1 の正規分布に従う確率変数の実現値として、$\min(1+\lfloor|\alpha|\rfloor,|A|)$によって定めた。$\lfloor\alpha\rfloor$は$\alpha$を越えない最大の整数である。文字列$s^{(k)}$の生起確率$p(s^{(k)})$は$\sum_{k=1}^{N} p(s^{(k)}) = 1$を満たすような一様乱数によって定めた。$p(s)$、$|s^{(k)}|$、$N$のそれぞれの場合について、$N$本の文字列の集合を 10 回生成し、Hayashida ら[2]のプログラム（ILPTri）と、本研究で定式化したプログラム（ILPMat）に同じ文字列を入力し、解を求めるまでの時間を記録した。整数線形計画問題を解くソフトウェア GLPK（version4.65）を使用し、Intel Xeon CPU E5-2687w v4 @ 3.00GHz、64GB 計算機で実行した。

4．実験結果

入力文字列 2 本、文字数 3～4 文字の条件において、距離が最小となる中央文字列が出力されていることを確認した。

表 1 にN=3 の条件で平均実行時間と標準偏差を示す。ILPTri の N≥4 では、500000 秒経過しても 10 回の試行が完了しなかった。$|s^{(k)}|$=2 では、ILPMat の平均実行時間は ILPTri よりも小さくなったが、$|s^{(k)}|$=3 では大きくなっている。一方、標準偏差は$|s^{(k)}| = 2,3$ともに ILPMat が小さくなっているため、計算時間が安定していることがわかる。ILPTri の方が時間短縮できているときに、入力文字列中に同じ文字列が含まれていることが多かった。

表 1　N=3 のときの平均実行時間と標準偏差

長さ$\lvert s^{(k)}\rvert$	ILPTri [s][2]	ILPMat [s]
2	0.013±0.009	0.011± 0.005
3	0.174±0.173	0.219± 0.058
4	————	3.741± 1.309
5	————	287.241± 720.815
6	————	1178.071±2654.057

$|s^{(k)}| \ge 4$では ILPTri の計算は完了していない。したがって ILPMat の計算時間は短くなっている。

5．考察

文字列の長さが長くなると ILPTri より時間短縮になるのは、中央文字列の解候補を削減する制約式があるからだと考える。N=3,n_k＝3 のとき ILPTri の方が早い。それは、$m_{19}^{(k)}$＝1 のときの解候補のみ削除するため、解候補の削除の制約式の効果があまりないからである。N=3,n_k＝5 のとき ILPMat の方が早いのは、$m_{1,11}^{(k)} = m_{1,12}^{(k)}$=1 のときの解候補を削除することができるからである。

標準偏差が大きくなった要因を考える。縦に同じ文字が入力された場合はマッチ候補を見つけやすいが、ない場合は見つけにくい。入力文字列によってマッチ候補の探索時間の差があるため、標準偏差が大きくなった。

参考文献

1)T.Kohonen, Median String, Pattern Recognition Letters, Vol. 3, pp.309-313, (1985)

2)M.Hayashida, H.Koyano, Integer Linear Programming Approach to Median and Center Strings for a Probability Distribution on a set of Strings, Biomedical Engineering Systems and Technologies, pp.108-121, (2016)