社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

短い部分文字列のミスマッチトレランスを高速計算するアルゴリズム

宇野　毅明†

† 国立情報学研究所, 〒 101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: †uno@nii.jp

あらまし　ゲノムの各部分列に対し, 他の部分列とのハミング距離の最小値をミスマッチトレランスという. ミスマッチトレランスは部分列がゲノムの中でどれくらい固有かを示す指標であり, 遺伝病の診断に応用がある. 固定した長さ部分列全てのミスマッチトレランスを計算する既存のアルゴリズムは, ゲノムの長さの 2 乗の計算時間がかかり, ヒトゲノムのような巨大な入力に対しては, 現実的な時間内に終了しない. 本稿では, ミスマッチトレランスを計算する線形時間アルゴリズムを提案する. また, 実際の計算時間を短縮する手法についても議論する. 計算実験により, 1 千万を超える長さのゲノムに対しても実用的な時間で計算が終了することを示す.

キーワード　類似部分列, マイクロアレイ

An Efficient Algorithm for Computing Mismatch Tolerance of Substrings

Takeaki UNO†

† National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, JAPAN, 101-8430
E-mail: †uno@nii.jp

Abstract　For a given substring S of length n of genome sequence G, the mismatch tolerance of S is the minimum humming distance to other substrings of G. The mismatch tolerance represents how much unique S is, and has an application to diagnosis of diseases. The existing algorithms for computing mismatch tolerance for all substrings of length n takes time of square of the length of genome, thus it is not efficient for real huge input such as human genomes. In this paper, we propose a linear time algorithm for this problem. Moreover, we discuss the technique for reducing practical computation time. By computational experiments, we show that our algorithm terminates in efficiently short time for large input with over 10 milion letters.

Key words　similar substring, micro array

1. はじめに

近年ヒトゲノムの解読が完了し, 人類は自らの遺伝情報を文字列情報の形で手に入れた. 今後はこの遺伝情報を解析し, 新薬の開発や遺伝病の治療が行われていくだろう. また, 他の生物のゲノム解読も着々と進んでおり, 遺伝子の発見や, 人類や種の起源を探る研究が進むと期待されている.

これらの研究で大事な働きをするのが, ゲノムの解析を効率よく行う各種ツールである. 例えばゲノムの中からある種の特徴のある部分を列挙したり, あるいはゲノムの部分列の類似度の計算, 及びある種の部分列と似たものを探し出すといった, 基礎的な文字列の問題を解く効率的なアルゴリズムである. これらの問題は, 計算理論上は非常に簡単な部類に属し, ほとんどの問題がそれほど凝った工夫なしに多項式時間で解ける. しかし, ゲノムを入力とした計算は, その入力の大きさゆえに時間がかかるため, 単に多項式時間というだけでは, 効率的なアルゴリズムの保証とはなりえない. ヒトゲノムはおよそ 30 億文字からなるので, 例えば入力の 2 乗ステップの計算を行うアルゴリズムは, おおよそ 900 京 (京は兆の 1 万倍) ステップの計算を行うことになる. これでは, たとえ 1 秒間に 100 億ステップの計算ができるコンピュータでも, 30 年程度の時間がかかる. これら基礎的なゲノム解析の問題に対しては, より次数の小さな多項式時間の, あるいは実用上高速なアルゴリズムの開発が重要なのである.

今回, 本論文ではゲノム部分列のミスマッチトレランスを計算する問題を扱う. ゲノム G の部分列 S のミスマッチトレランスとは, S と, S と長さが等しい他の G の部分列とのハミング距離の最小値のことであり, L がゲノムの中でどの程度固有であるかを表す指標である. ミスマッチトレランスが大きい場合, ゲノムの中に類似する部分列は存在しない. よって, そのような部分列は, たとえ部分列やゲノム自体に数文字の読み間違いが存在しても, ゲノムの中の出現位置が唯一的に定まるため,

マイクロアレイを用いた遺伝病検査の分野に応用がある.

ある人物が遺伝的な要因による病気を持つかどうかを調べるには, その人物のゲノムが, 病気の原因となる遺伝子, つまりはある特定の部分列を含むかどうかを検査する. この検査は, 通常マイクロアレイを用いて行う. マイクロアレイは, 指定された部分列がゲノムに含まれるかどうかを検査できるが, どのような部分列でも調べられるわけではなく, 20 数文字程度の部分列の存在判定しかできない. しかし, 一般に病気の原因となる遺伝子はそれほど短くなく, 少なくとも 20 文字程度に収まるものではない. そのため, 遺伝子を 20 数文字ずつに分割し, 各々がゲノムに含まれるかどうかを調べることにより, ゲノムが遺伝子を含むかどうかを検査する, という方法が一般的に行われている. しかし, この方法にも弱点がある. 分割してできた部分列がゲノムの他の部分列とたまたま一致してしまうと, 実際にはその遺伝子を含まなくても, 含んでいると判断されてしまう. また, 正確に他の部分と一致しなくとも, 少しの変異によって一致してしまうような部分列が存在する, つまりは似た部分列が存在していても, そのような間違いが起こる危険はある. 部分列への分割の仕方は複数あるので, なるべくそのようなことが起きないような分割を行うことが重要であり, そのためには, ゲノムの各部分列がどの程度固有であるか（どの程度似た部分列が存在するか）を知る必要がある. そのために, ミスマッチトレランス計算が必要なのである.

ゲノム G の各部分列に対するミスマッチトレランスは, 簡単な動的計画法アルゴリズムを用いて, $\mathrm{O}(|G|^2)$ 時間で計算できる. ここで $|G|$ はゲノム G の長さである. しかし, 上記の通り, ゲノムを入力とする 2 乗時間アルゴリズムは実用的な時間では終了しない. そのため, 与えられたゲノム G と短い文字列 S (S は G の部分列とは限らない) のミスマッチトレランス $mt(S,G)$ を計算するアルゴリズムが研究の主流となっている. 例えば Abrahamson [1] は, $mt(S,G)$ を $\mathrm{O}(|G|(|S|\log|S|)^{1/2})$ 時間で計算するアルゴリズムを提案している. また, ハミング距離の計算に誤差を許し, 確率 δ で計算誤差が $(1-\epsilon)$ から $(1+\epsilon)$ に収まるようなアルゴリズムが Feigenbaum, Kannan, Strauss and Viswanathan によって提案されている. これを用いると, 多少計算時間は短くなり, $\mathrm{O}(\log|S|\log(1/\delta)/2)$ となる [2]. しかし, これらのアルゴリズムは, いくつもの S に対してミスマッチトレランスを計算しようとすると, 多大な時間を要することになる.

また, 山田・森下 [3] は $mt(S,G)$ の下界を求めるアルゴリズムを提案している. このアルゴリズムは, 最初に G を入力して前処理を行った後, 以後与えられた S に対して $\mathrm{O}(|S|L)$ 時間で $mt(S,G)$ の下界を求めることができる. ここで L は $|S|$ より小さな定数である.

2. この研究の貢献

本稿では, ゲノム G と定数 n を入力し, G の長さ n の各部分列のミスマッチトレランスを正確に計算する線形時間アルゴリズムを提案する. 計算時間は $\mathrm{O}(|G|n2^n)$ 時間であり, n に対しては指数時間であるが, ゲノムの長さに対しては線形時間であるため, 長いゲノムの短い部分列に対するミスマッチトレランスを高速に計算できる. このアルゴリズムは, ミスマッチトレランスが d 以下であるかどうかを調べる問題を 0 から n までの d について解く, という方法を用いている. 本稿の中心の結果は, この子問題を $\mathrm{O}(|G|(n-d)_nC_d)$ 時間で解くアルゴリズムを提案したことにある. さらに, 実用上の計算時間を短縮する方法の提案も行う. このアルゴリズムは計算機に実装済みである. 本稿にも計算実験の結果を記載し, 実際の計算時間が最悪の計算時間よりも大幅に短くなることを示す.

本稿のアルゴリズムは, 少々の変更を加えることにより様々な問題を解くことができる. 本稿では, それらの中から, 複数のゲノムの比較を行い, 各ゲノムの部分列に対して, 他のゲノムに対するミスマッチトレランスを計算する問題, ミスマッチトレランスをハミング距離でなく一般のエディット距離で定義した問題, ハミング距離が k 以下の部分列を全て列挙する問題, 山田・森下 [3] が提案した拡張ミスマッチトレランスを計算する問題への応用を解説する.

3. 線形時間アルゴリズム

文字列 S に対して, S の i 文字目を $S[i]$ と表記する. また, S の長さを $|S|$ とする. 長さが等しい 2 つの文字列 S_1 と S_2 に対して, S_1 と S_2 の**ハミング距離** $S_1[i] \neq S_2[i]$ となるような i の個数で定義し, $d_H(S_1,S_2)$ と表記する. 文字列 G の部分文字列 S の**ミスマッチトレランス** $mt(S,G)$ を S と G の他の部分列とのハミング距離の最小値で定義する. つまりは, G の長さ $|S|$ の部分列の集合を $G^{|S|}$ とすると

$$mt(S,G) = \min_{S' \in G_{|S|}} \{d_H(S,S')\}$$

である.

この節では, 固定した d と n に対して, 文字列 G の長さ n の各部分列 S のミスマッチトレランス $mt(S,G)$ が d 以下かどうかを判定する問題を解く線形時間アルゴリズムを解説する. $mt(S,G)$ を正確に計算するには, 0 から n までの各 d に対してこの問題を解けばよい.

アルゴリズムのアイディアは, 以下で示す補題から出てきたものである. 添え字集合 I に対して, 部分列上の同値関係 $=_I$ を,

$$S_1 =_I S_2 \Leftrightarrow S_1[i] = S_2[i] \text{ for any } i \in I$$

で定義する. また, $\mathcal{C}(n,m)$ を, 1 から n の添え字をちょうど m 個選んで得られる添え字集合全ての集合とする. このとき, 以下の補題が成り立つ.

［Lemma 1］ $mt(S,G) \leqq d$ であれば, またそのときに限り, ある $I \in \mathcal{C}(|S|, |S|-d)$ とある部分列 $S' \neq S$ が存在して $S =_I S'$ が成り立つ. ∎

証明は, 明らかなので省略する. この補題より, $mt(S,G) \leqq d$ かどうかを計算するには, 全ての $I \in \mathcal{C}(|S|, |S|-d)$ について, $S =_I S'$ となる S' が存在するかどうか調べればよいということがわかる. 与えられた I に対して, G^n を $=_I$ での同値類に

分類したとする．ある同値類が 1 つしか部分列を含まなければ, その部分列 S に対して, $S =_I S'$ となる S' は存在しない．逆に, 同値類が複数の部分列を含めば, それら部分列 S, S' は, $S =_I S'$ を満たす．よって, 全ての $I \in \mathcal{C}(|S|, |S| - d)$ について, G^n を $=_I$ での同値類で分類する, という作業を行うことにより, ミスマッチトレランスが d 以下となるかどうかを調べることができる．この, $\mathcal{C}(|S|, |S| - d)$ の全ての添え字集合について, G^n を同値類に分類する問題を $P(G^n, \mathcal{C}(|S|, |S| - d))$ と表記する．

$P(G^n, \mathcal{C}(|S|, |S| - d))$ を解くには G^n の部分列を 各 $i \in I$ 文字目をキーにしてソートし, 分類すれば良い．ゲノムは 4 種類のアルファベットしか含まないため, この作業は $O(|G^n||I|)$ 時間でできる．よって, 以下の定理を得る．

［Theorem 1］ ゲノム G の長さ n の各部分列 S に対して $mt(S, G) \leqq d$ が成り立つかどうかを $O(|G^n|(n-d)_nC_d)$ 時間で調べられる．∎

この手法は, $|G^n|$ に対して線形のメモリしか使用しないこと, アルゴリズムが入力する添え字集合 $\mathcal{C}(|S|, |S| - d)$ は, 陰的な表現を用いて定数サイズのメモリしか使用しないことを注意しておく．

4. 高 速 化

前節のアルゴリズムは $|G^n|$ に対しては線形時間であるが, n と d に対しては指数時間である．そのため, $n = 10$, $d = 2$ 程度であれば非常に高速であるが, n と d が大きくなるにつれ, 遅くなる．例えば G^n が 1 千万程度, $n = 30, d = 5$ とすると, $|G^n|(n-d)_nC_d$ と $|G^n|^2$ は同程度の大きさになる．現実問題に応用するためには, 多少大きめの n や d に対しても短時間で計算が終了することが望ましい．この節では, 実験的な計算時間を短縮する方法について解説する．

本稿では, 4 つの高速化を提案する．1 つ目は再帰的な分類によるもの, 2 つ目は分類方法を変更するもの, 3 つ目は再起の途中で限定操作を行うもの, 4 つ目は無駄な計算を省略するものである．以下で, これら 4 つの方法を 1 つずつ解説する．

まず 1 つ目の再帰的な分類による高速化を解説する．$\mathcal{C}(n, d)$ の添え字集合を 1 を含むものと含まないものに分類し, 前者を $\mathcal{C}_1$, 後者を $\mathcal{C}_2$ とする．ここで, $J = \{1\}$ とすると, 任意の $I \in \mathcal{C}_1$ に対する同値類は $=_J$ での同値類に含まれる．そこで, G^n を $=_J$ の同値類 $L_1, .., L_4$ に分類し, 各 L_j について子問題 $P(L_j, \mathcal{C}_1)$ を再帰的に解くことにより問題を解けば, 1 文字目の分類作業を 1 回しかしないですむ．子問題もさらに再帰的に解くことによって, 2 文字目, 3 文字目に関する分類作業も, 省略することができる．

もし, $=_J$ の同値類の 1 つ L_j が $|L_j| = 1$ であれば, 以後, 再帰的に分類を繰り返しても, 同値類の大きさは 1 のまま変化しない．よって, このような L_j に関する再起呼び出しは必要ない．また, $|L_j| = 2$ であれば, 再起呼び出しをせずとも, 直接 L_j に含まれる 2 つの部分列を比較すればよい．このような限定操作, あるいは直接的に解く操作により, 再帰が深くなるにつれて計算する必要がある部分列の数はしだいに小さくなり, 計算時間の短縮が見込まれる．この高速化を適用したアルゴリズムを以下に記述する．

アルゴリズムの実行前, 任意の S に対して $mt(S, G) > d$ とする
ALGORITHM method1 ($\mathcal{S}$:部分列の集合, $\mathcal{I}$: 添え字集合の族, i:添え字)
1. $|\mathcal{S}| = 1$ ならば終了
2. $|\mathcal{S}| = 2$ ならば, $S, S' \in \mathcal{S}$ のハミング距離を直接計算し, 終了
3. $i = n$ ならば, $\mathcal{S}$ の各部分列 S に対して, $mt(S, G) \leqq d$ とし, 終了
4. $\mathcal{I}$ を i を含むものと含まないものに分割し, それぞれを $\mathcal{I}_1, \mathcal{I}_2$ とする
5. method1 ($\mathcal{S}$, $\mathcal{I}_2$, $i+1$) を呼び出す
6. $\mathcal{S}$ の部分列を, i 文字目をキーとして同値類 $\mathcal{S}_1, ..., \mathcal{S}_4$ に分類する
7. 各 $\mathcal{S}_j$ に対して, **method1** ($\mathcal{S}_j$, $\mathcal{I}_1$, $i+1$) を呼び出す

2 つ目の高速化は, 上記の再帰呼び出し型のアルゴリズムの, 子問題の作り方を変更するものである． まず最初, 添え字集合 I を $\{1, ..., n\}$ とする． そして, I を 2 つの集合 $I_1 = \{1, ..., \lceil n/2 \rceil\}, I_2 = \{\lceil n/2 \rceil + 1, ..., n\}$ の 2 つに分割する．すると, $\mathcal{C}(n, d)$ の添え字集合は, I_1 の添え字をいくつ含むかによって $\mathcal{C}_0, ..., \mathcal{C}_d$ の $d+1$ 種類に分類される．各 $\mathcal{C}_j$ について子問題 $P(G^n, \mathcal{C}_j)$ を解けば, 元の問題が解ける．

各 $\mathcal{C}_j$ に関する子問題は, $j = 0$ であれば I_1 の添え字について分類を行った後, 各同値類 L について $P(L, \mathcal{C}_j \cap I_2)$ を解けばよい．ただしここで

$$\mathcal{C}_j \cap I_2 = \{I \cap I_2 | I \in \mathcal{C}_j\}$$

とする．同様に $j = \lceil n/2 \rceil$ であるときも, I_2 の添え字で分類した後, 各同値類について子問題を解けばよい．$j \neq 0, \lceil n/2 \rceil$ である場合は, I_1 を同様に分割することにより, $\mathcal{C}_j$ を再帰的に分割できる．

このように, 再帰的に分割を行った場合の, 各子問題の計算時間の違いについて考察しよう．子問題 $P(G^n, \mathcal{C}_j)$ を再帰的に分割し, I_1 に含まれる添え字について全て分類が終了する深さの反復の数を R_j, それら反復が入力する部分列の数の平均を Q としよう．今, $\mathcal{C}_j$ の添え字集合が I_1 の添え字を $\lceil n/2 \rceil - j$ 個含むとすると, 実際の計算では, Q と R_j は j が大きくなれば増大する傾向があると考えられる．つまりは, $P(G^n, \mathcal{C}_0)$ が最も速く解け, $P(G^n, \mathcal{C}_d)$ が最も時間がかかると考えられる．そこで, $j > d/2$ なる j については, 再帰呼び出しで子問題を解く際に, I_2 の添え字を先に固定して, その後 I_1 の添え字を固定するようにすれば $P(G^n, \mathcal{C}_{d-j})$ と同程度の時間で解けるようになることが期待できる．さらに子問題での分割でも同様な操作を行えば, 子問題を解く時間自体も改善されると期待される．この高速化を行ったアルゴリズムを以下に示す．

ALGORITHM method2 ($\mathcal{S}$:部分列の集合, $\mathcal{I}d$: 添え字集合と自然数の組の族)
1. $|\mathcal{S}| \leqq 2$ か $\mathcal{I}d = \emptyset$ ならば直接ハミング距離を計算し, 終了

2. $\mathcal{I}d$ から添え字集合 I と自然数 d の組を取り出す
3. $d = |I|$ ならば **method2** $(\mathcal{S}, \mathcal{I} \setminus \{(I,d)\})$ を呼び出し, 終了
4. $d = 0$ ならば, $\mathcal{S}$ の部分列を全ての $i \in I$ 番目の文字で
同値類 $\mathcal{S}_1, ..., \mathcal{S}_m$ に分類し, 各 $\mathcal{S}_j$ に対して **method2**
$(\mathcal{S}_j, \mathcal{I} \setminus \{(I,d)\})$ を呼び出し, 終了
5. I をほぼ同じ大きさの添え字集合 I_1 と I_2 に分割する
6. 0 から d までの各 d' について
method2$(\mathcal{S}, \mathcal{I} \setminus \{(I,d)\} \cup \{(I_1,d'),(I_2,d-d')\})$ を呼び出す

3 つ目の高速化は, 各反復で, 入力サイズが小さかったときにはさらなる再帰は行わず直接問題を解くというものである. ある反復が問題 $P(D,\mathcal{C})$ を入力したとする. このとき, 直接的に問題を解いたときにかかる計算時間と再帰呼び出し型の計算で解いた場合の計算時間をおおまかに算定し, 直接解いた方が有利だと思われたら, 各部分列の組のハミング距離を求め, 直接的に問題を解く. 本稿のアルゴリズムでは, 両者を $|D|^2$ と, $|D|(2^d)$ で算定した.

4 つ目の高速化は, すでにミスマッチトレランスが計算できている部分についての再計算を省略するものである. ある反復が問題 $P(D,\mathcal{C})$ を入力したとする. このとき, もし D の任意の部分列についてミスマッチトレランスが d 以下であることが分かっているならば, この反復は直ちに終了できる. また, この問題 $P(D,\mathcal{C})$ を直接的に解く場合, ミスマッチトレランスが d 以下であることが分かっている部分列同士は比較する必要は無い.

5. 拡　　張

本稿のアルゴリズムは, 添え字集合が導く同値関係を用いて部分列を分類するという手法でミスマッチトレランスを計算している. この, 分類するという基本的な構造を用いると, いくつかの他の類似した問題を解くことができる. この節では, そのような問題をいくつか提示し, 本稿のアルゴリズムの構造を用いてそれらの問題を解く方法を解説する.

5.1 複数のゲノム

複数のゲノムに対して, 各部分列の他のゲノムに対するミスマッチトレランスを計算する問題を考える. 正確には, 2 つのゲノム G_1, G_2 と定数 n が与えられたとき, G_i の長さ n の各部分列 S に対して, $mt(S, G_{1-i})$ を計算する問題である.

この問題を解くには, G_1^n と G_2^n をあわせた部分列集合に対して, 上記のアルゴリズムを適用すればよい. ただし, そのまま適用したのでは, 不具合が発生する. G_i の部分列 S に対して, G_i の他の部分列 S' で $d_H(S,S') < mt(S, G_{1-i})$ となるものが存在した場合, ミスマッチトレランスの値が間違って報告されてしまう. それを回避するためには, アルゴリズムの実行中, 添え字集合 I に関する同値類 R に対して, R が G_1 と G_2 の両者の部分列を含むときのみ, R 内の部分列のミスマッチトレランス値が d 以下であると見なせばよい.

5.2 ハミング距離が小さい部分列組の列挙

与えられたゲノムのハミング距離が短い部分列の組を全て見つける問題を考える. 正確には, ゲノム G と定数 n, d が与えられたとき, G の長さ n の部分列 S と S' の組で, $d_H(S,S') \leqq d$ となるものを全て列挙する問題である.

この問題は, 本稿のアルゴリズムを実行し, 得られる各同値類 R に対して, R に含まれる部分列の組を全て出力すれば解ける. しかし, ハミング距離が d 未満の組は, $\mathcal{C}(n,d)$ の複数の添え字集合に対して同値関係が成り立つため, 複数回出力されてしまう. この点は, 部分列の組をメモリに蓄え, 重複を除けば解決できるが, そのためには, 多大なるメモリを要する可能性がある. そこで, ここでは, メモリを使わずに重複を回避する方法を解説する.

本節の方法は, d' を 0 から d まで 1 つずつ大きくし, 各 d' について $d_H(S,S') = d'$ となる部分列の組を列挙する問題を解く, というものである. そのためには, アルゴリズム実行中に得られる同値類 R に対して, R に含まれる, ハミング距離がちょうど d' の組のみを出力し, ハミング距離が d' 未満の組は出力しないようにする. ハミング距離がちょうど　d' である部分列の組は, ちょうど 1 つの添え字集合 $I \in \mathcal{C}(n,d')$ に対してのみ同値関係を満たすので, 上記の方法を用いて出力の重複を避けられる.

添え字集合 $I \in \mathcal{C}(n,d')$ に対して同値関係を満たす部分列 S と S' に対して, S と S' のハミング距離が $d_H(S,S') = d'$ を満たすかどうかを調べるには, I に含まれない n 以下の各添え字 i について $S[i] = S'[i]$ が満たされるかどうかを調べればよい.

ただし, S か S' のどちらかのミスマッチトレランスが d' 以上であることがわかっているならば, $d_H(S,S') < d'$ であることはないので, ただちに $d_H(S,S') = d'$ と結論付けられる.

5.3 拡張ミスマッチトレランス

拡張ミスマッチトレランスとは, 山田・森下 [3] によって提案された類似度の尺度である. ゲノム G と 定数 n と k が与えられたとき, G の部分列 S に対して, S の拡張ミスマッチトレランス $mt_k(S,G)$ は, S と k 番目にハミング距離が短い G の部分列とのハミング距離で定義される. より正確には, $mt_k(S,G)$ は, S 以外の G の部分列で, ハミング距離が $d-1$ 以下のものが $k-1$ 個以下, d 以下のものが k 個以上となるような d で定められる.

山田・森下は, 与えられたゲノム G と部分列 S に対して拡張ミスマッチトレランスの下界を高速に計算するアルゴリズムを提案している. しかし, 全ての部分列に対して拡張ミスマッチトレランスを計算するには, $\mathrm{O}(|G|^2)$ 程度の時間がかかる.

本稿では, 拡張ミスマッチトレランスを計算するために, 上記のハミング距離が短い部分列の組の列挙を用いた方法を提案する. 上記アルゴリズムを $d = 0$ から順に d を 1 ずつ大きくしていくと, 部分列の組をハミング距離の短い順に列挙することができる. 各部分列 S に対し, ちょうど k 番目に出力された S を含む組 (S,S') のハミング距離 $d_H(S,S')$ が S の拡張ミスマッチトレランスとなる.

高速化の節で述べた 4 つ目の改良と同じく, 拡張ミスマッチトレランスの計算が終了した部分列だけを入力した反復はただちに終了することができる. これにより, 実際の計算での高速化も期待できる.

5.4 一般の距離

一般にゲノムの部分列を比較する場合には, エディット距離, あるいはエディット距離にある種のスコアをつけたものが用いられる. エディット距離は, 文字の変更, 1 文字挿入, 1 文字削除という 3 種類の文字列の操作にこれは, ある部分列 S と S' が与えられたとき, S から S' へ 3 種類の文字列操作で変換を行う最もコストの安い方法を求め, そのコストを S と S' のエディット距離と定義するものである. ここでは, 本稿のアルゴリズムでエディット距離を扱う方法を解説する.

$I = (i_1, ..., i_{n-d})$ と $J = (j_1, ..., j_{n-d})$ を, 2 つの添え字列 $(1, ..., n)$ から以下の操作を d 回行って得られる添え字列の組とする.

・片方の添え字列から添え字 i を, もう片方から最後の添え字を取り除く
・両方の添え字列から添え字 i を取り除く

直感的には, 1 つ目が挿入／削除に対応し, 2 つ目が文字の変更に対応する. このとき, 長さ n の部分列 S 上の 2 項関係 $\rightarrow_{I,J}$ を,

$$S \rightarrow_{I,J} S' \Leftrightarrow S[i_g] = S'[j_g] \text{ for any } 1 \leqq g \leqq n - d$$

で定義する. このような添え字組の集合を $\mathcal{D}(n, d)$ とする. このとき, 以下の補題が成り立つ.

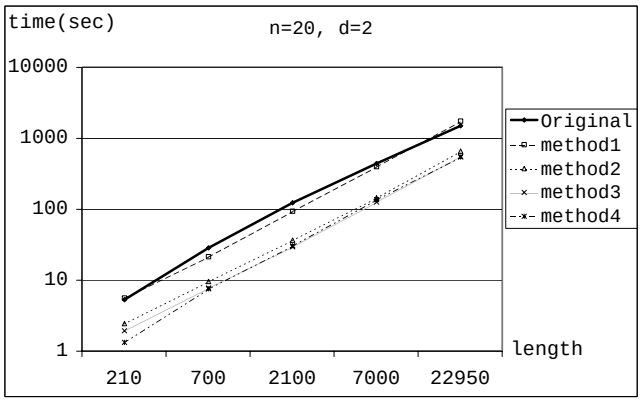
[Lemma 2] ゲノム G の中に部分列 S とエディット距離が d 以下の部分列 S' が存在するとき, またそのときに限り, ある $(I, J) \in \mathcal{D}(n, d)$ に対して $S \rightarrow_{I,J} S'$ が成り立つ. ∎

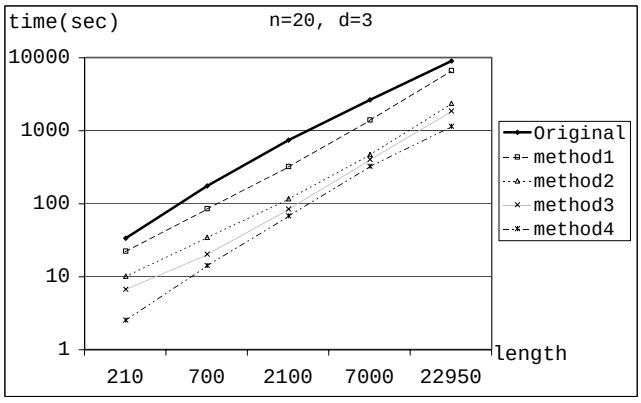
これは, ミスマッチトレランス計算における補題 1 に対応する. よって, 全ての $(I, J) \in \mathcal{D}(n, d)$ に対して, G_n を $=_I$ および $=_J$ の同値類に分類することで, エディット距離を用いた計算を行うことができる.

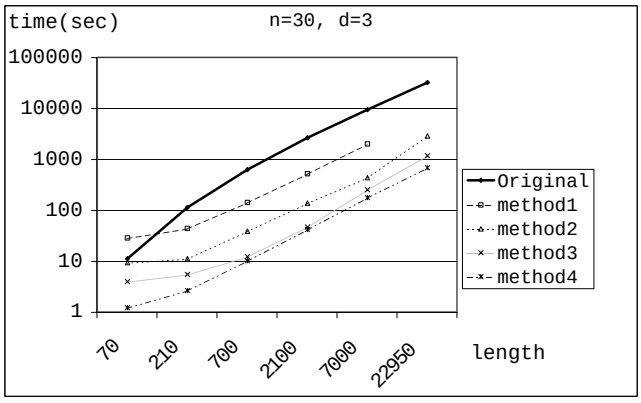
6. 計 算 実 験

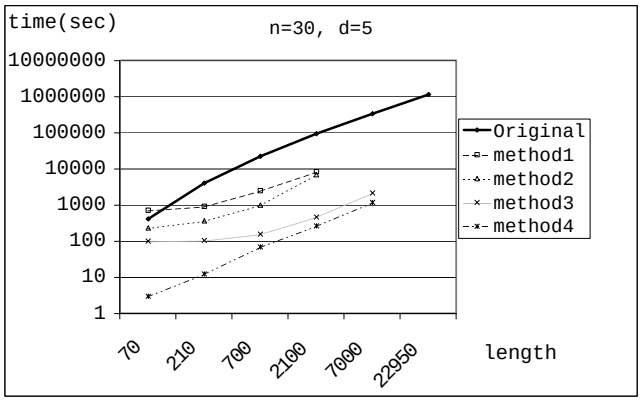
本研究では, 4 節で述べた高速化を行ったミスマッチトレランス計算アルゴリズムを計算機に実装した. この節では, 実際にゲノム列に対して計算を行った結果を示す. ソースコードは C で書き, コンパイルには gcc を用いた. 実験に用いたコンピュータは PentiumIII 750MHz のノート PC, OS は cygwin (windows 上で動作する linux 環境) である. 実験には, ヒトゲノムの Y 染色体, およびそれから切り出した部分列を用いた. Y 染色体は長さが短めであり, ノートパソコンの環境でも十分計算ができる, というのが Y 染色体を選んだ理由である.

実験は $n = 20, d = 2, n = 20, d = 3, n = 30, d = 3, n = 30, d = 5, n = 40, d = 4, n = 40, d = 7$ の 6 種類のパラメータについて行った. 以下に実験結果のグラフを示す. X 軸がゲノムの長さ, Y 軸が計算時間 (秒) であり, それぞれの軸は対数目盛りになっている. original はソートを ${}_nC_d$ 回行うのにかかる時間, method i が高速化手法 1 から i を適用したアルゴリズムの計算時間である. original は高速化を行っていない基本アルゴリズムの計算時間の下界であり, 実際の基本アルゴリズムの計算時間よりは大幅に短いであろうことを注意しておく.

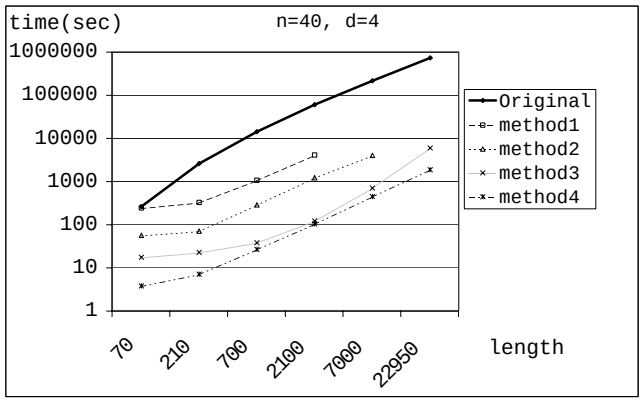


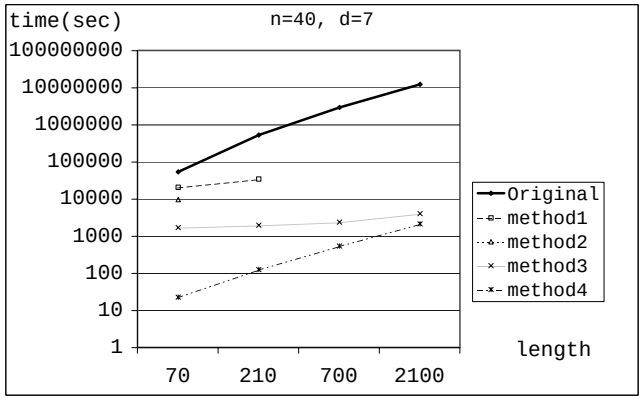






実験結果より, 4 つの高速化手法がそれぞれ現実の計算時間短縮に貢献していることが分かる. 全般的に高速化手法 1 の改善は大きく, d が小さいときは 2 つ目の高速化手法が, また d が





大きいときには 3 つ目の高速化手法が, そしてゲノムの長さが短いときには 4 つ目の高速化手法が大きく貢献していることが分かる.

なお, このアルゴリズムを 0 から n までの各 d に対して適用し, ミスマッチトレランスを求める時間は, 例えば $n = 20$ の場合, $n = 20, d = 2$ の計算時間のほぼ 10 倍程度であった.

7. ま と め

本稿では, ゲノムの各部分列のミスマッチトレランスを正確に計算する, ゲノムの長さに対する線形時間アルゴリズムを提案した. また, 実際の計算が高速化される改良をいくつか議論し, それらが有効であることを計算実験によって確認した. 本稿のアルゴリズムにより, ヒトゲノムに対するミスマッチトレランスの計算を行うことも可能になった. また, いくつかの他の問題に対する本稿のアルゴリズムの適用法についても論じた. 本稿のアルゴリズムは部分列の長さが長くなるにつれ計算時間が長くなるため, 今後はより長い部分列に対するミスマッチトレランスの計算を高速に行うような改良法の研究が必要であろう.

文 献

[1] K. Abrahamson, “Generalized string matching,” SIAM Jour-nalon Computing, **16(6)**, pp. 1039 - 1051, 1987.

[2] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, “An approximate l1-difference algorithm for massive data streams,” In Proc. of the 40th IEEE Annual Symposium on Foundations of Computer Science (FOCS1999), 1999.

[3] Tomoyuki Yamada and Shinichi Morishita, “Computing Highly Specific and Mismatch Tolerant Oligomers Efficiently,” Bioinformatics Conference 2003.