メタ文字を含む文字列に対する Vantage-Point 木を用いた類似文字列検索

An Approximate String Matching Method which Uses Vantage-Point Tree for Strings Containing Meta-Characters

森川　浩司　Morikawa Koji
高梨　勝敏　Takanashi Katsutoshi
宗形　聡　Munakata Satoshi

名称が似ている別の薬を投薬してしまうミスの防止が課題になっている。ミスを防止するためには名称の類似性だけでなく投与量の観点でのチェックも重要である。このように医療・健康福祉分野では数値範囲を伴う項目に対する誤入力防止の技術が求められている。本報告では，複数の数値選択肢からひとつ選択することを表すメタ文字を含む文字列群から，それらの文字列とは一致しない，メタ文字を含まない文字列と類似した文字列を，編集距離と Vantage-Point 木を用いて高速に検索する方法を提案する。これまでの研究で行った，数字文字列を単位文字とする編集距離の定義を用いて Vantage-Point 木を構築して検索を行うと，検索漏れが発生するという課題があった。この課題に対して，Vantage-Point 木の構築時は編集距離として Hausdorff 距離を用い，検索時は数字文字列を単位文字とする距離を用いることでこの課題が解決できることがわかった。この解決策では，距離をインデックスとする木構造が持つ枝刈りのメリットが一部失われるが，それでもなお木構造を用いない検索よりも高速であることをデータで検証した。

１．はじめに

医療・健康福祉分野では名称が似ている別の薬を投薬してしまうミスの防止が課題になっている。この防止のための施策としては，名称の類似度の定量的な評価に基づく名称チェックだけでなく，投与量など量の観点といった名称と名称以外の観点を合わせたチェックも重要である。このように，名称が類似していて異なる数値範囲を持つ多数の項目からすばやく類似項目を検索・提示する技術は今後重要になってくる。

有限個の数値選択肢から数値をひとつ選択する表記法として，例えば 1 から 100 の範囲内で 0.5 刻みの数値からひとつを選択する場合は「{1..100(0.5)}」と書いたり，あるいは 2，4，7，10 の中からひとつを指定する場合は「{2,4,7,10}」と書いたりすることができる。このとき「{」「}」をメタ文字と呼び，このメタ文字を使って，複数の数値選択肢から数値をひとつ選択することを表せる。以降，数値選択肢からのひとつの数値選択を表すメタ文字を数値選択メタ文字と呼び，数値選択メタ文字を含む文字列を数値選択メタ文字列と呼ぶ。メタ文字はその意味を持たせられれば文字としては「{」「}」でなくても，例えば「<」「>」や「[」「]」などでもよい。また数値選択肢は「{1..100(0.5)}」を「{1～100|0.5}」と書いたり，「{2,4,7,10}」を「{2/4/7/10}」と書いたりすることもできる。今回の研究開発対象は，その表記法は問わず，複数の有限な離散数値選択肢からひとつを選択する表現を含む文字列に対する類似文字列検索である。以降では便宜上，数字選択メタ文字表記を「{,,,}」の形式で表す。

メタ文字を含む文字列に対する類似文字列の検索技術としては，メタ文字表記が正規表現である場合には先行研究 [1), 2)] が豊富であるが，今回のような数値選択メタ文字に関しては見出すことができない。

数値選択メタ文字列は数値選択メタ文字を含まない文字列の集合と見ることができる。例えば，数値選択メタ文字列「ABC{1,20,300}」は数値選択メタ文字を含まない「ABC1」・「ABC20」・「ABC300」の各文字列を要素とする集合と見ることができる。そこで，数値選択メタ

文字列を数値選択メタ文字を含まない文字列に分解すれば，個々の文字列に対して既存の類似文字列検索技術 [3] を適用することで数値選択メタ文字に対する類似文字列検索を行うことができる。しかし数値選択肢の数が多い状況を想定すると，そのような分解を行えば検索対象文字列の数が激増することが予想される。そこで，数値選択メタ文字を含まない検索文字列と類似した数値選択メタ文字列を，上記のような分解をすることなく，かつ高速に抽出する技術が必要となる。

(株)日立ソリューションズ東日本はこれまで，数値選択メタ文字を含む文字列群に対する類似文字列検索技術の研究開発を行ってきた。その中で文字列の類似度指標として編集距離 [3, 4] を用い，数値選択メタ文字列に対する編集距離を定義した。編集距離のように類似度が距離の定義を満たす場合は距離をインデックスとする木構造を用いることで検索を高速化できる。そこで，距離をインデックスとする木構造の中で一般的な Vantage-Point 木 [5, 6]（以降，VP 木）の利用を試みた。しかし上記で定義した編集距離と VP 木を組み合わせると検索漏れが発生した。本報告では検索漏れを回避するための施策と，その施策を用いて VP 木検索を行うと VP 木を用いない場合より高速に検索漏れなく検索できることの検証について報告する。

２．数値選択メタ文字列に対する編集距離の定義

ここではこれまでの研究開発の中で行った，数値選択メタ文字列に対する編集距離の定義について述べる。

付録Ａに記した編集距離を数値選択メタ文字列に適用するために，編集距離の計算に関して以下のルールを新たに設定した。

編集距離計算ルール(1)

数値文字列は単位文字（1 文字）として扱う（以降，これを単位数値文字と呼ぶ）

このルールによって，数値選択メタ文字部分にある選択肢の数値は，どれも長さ 1 の単位文字となる。(なお，数値選択メタ文字部分以外の数値文字列も単位文字扱いとする。) 例えば{1,20,300}は「1」という単位文字と「20」という単位文字と「300」という単位文字の中から文字をひとつ選択することを表すことになる。これにともない以下のルールも設定した。

編集距離計算ルール(2)

数値選択メタ文字部分も単位文字として扱う

数値選択メタ文字は選択肢の単位数値文字の中からひとつ選ぶことを表しているので，編集距離ルール(1)からこのルールは自動的に導かれる。以上のルールから，例えば A{1,20,300}B45 を単位文字に分解する場合，「A」「{1,20,300}」「B」「45」となる。

単位数値文字という新しい単位文字を導入したので，これに対応して文字の操作とそのコストとして以下のルールを新たに設定した。

編集距離計算ルール(3)

単位数値文字 n と N個の単位数値文字からなる数値選択メタ文字{$n_1,\cdots,n_N$}との置換

- n が $n_1,\cdots,n_N$ のどれかと一致すればコスト 0,
- n が $n_1,\cdots,n_N$ のどれとも一致しなければコスト 1

なお，単位数値文字と他の単位文字種との置換および削除・挿入・転置に関しては，単位数値文字および数値選択メタ文字部分は単位文字であるので従来の定義の枠内で扱える。

以上のルールを設定すると，数値選択メタ文字を含まない文字列 s と，数値選択メタ文字列 T（これを数値選択メタ文字を含まない文字列 $t1,\cdots,tN$の集合とみる）との距離 $d(s, T)$ は点・集合間距離 $d(s, T) = \min_{(i)}\{d(s, t_i) \mid t_i \in T\}$ となる。したがって，s と Tとの間の距離については

- s が $t_1,\cdots t_N$のどれかに一致する場合，およびそのときに限り距離 0 となる
- 対称である： $d(s, T) = d(T, s) = \min_{(i)}\{ d(s, t_i) \mid t_i \in T\}$
- 三角不等式は，数値選択メタ文字を含まない文字列を u とするとき $d(s, T) \leqq d(s, u) + d(u, T)$

となる。

３．検索高速化のための VP 木利用と検索漏れ

ここでは，前節で定義した数値選択メタ文字列に対する編集距離を用いての，付録 B に記した VP 木の構築および検索と，そこで発生した検索漏れという課題について述べる。

まず，VP 木の構築には数値選択メタ文字列間の編集距離の定義が必要であるので，編集距離計算ルールとして以下も設定する。

編集距離計算ルール(4)

M 個の数値文字からなる数値選択メタ文字{$m_1,\cdots,m_M$}と N個の数値文字からなる数値選択メタ文

字{n_1,…,n_N}との置換

- 単位数値文字の集合{ m_1,…,m_M}と{ n_1,…,n_N}とが集合として等しければコスト 0,
- そうでなければコスト 1

数値選択メタ文字を含む文字列と含まない文字列とが混在しているデータに対して以上のルールを適用して VP 木を構築して検索を行うと，検索漏れが発生した。理由は以下のとおりである。これまで設定したルールに則って作られた VP 木では，左右の枝刈りの前提となる距離の三角不等式は一般に成り立たない。数値選択メタ文字を含まない文字列を q, s, t, 数値選択メタ文字列を U, Vとするとき，これまでの定義から成り立つ距離の三角不等式は

$$d(q, s) \leqq d(q, t) + d(t, s)$$

および

$$d(q, U) \leqq d(q, s) + d(s, U)$$

だけであり，左側の枝刈り条件成立の前提となる三角不等式 $d(q, s) \leqq d(q, U) + d(U, s)$ や $d(q, U) \leqq d(q, V) + d(V, U)$，また右側の枝刈り条件成立の前提となる三角不等式 $d(U, V) \leqq d(U, q) + d(q, V)$は一般には成り立たないからである。また，距離の三角不等式が成り立っているノードでもその子ノードでは数値選択メタ文字列と数値選択メタ文字を含まない文字列が混在しているため，枝刈りの前提となる距離の三角不等式がそのノードのすべての子ノードで成り立つとは一般には言えない。図 1 にその例を示す。検索文字列 q（文字列：1A2B）と検索対象ノード s（文字列：10A20D，中央値 M=1），t（文字列：10C20D，中央値 M =2），U（文字列：{1,10}A{2,3}D）について，ノード sの左側の枝刈り条件が成り立っているが，ここで枝刈りを行うと抽出されるべき数値選択メタ文字列 Uを逃してしまう。

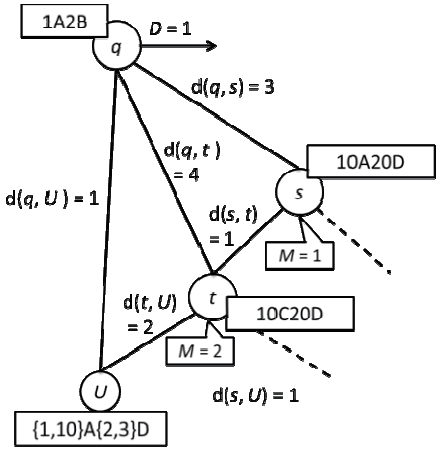

図 1　左側の枝刈り条件が成立しても枝刈りしてはいけない例

4．Hausdorff 距離の適用による検索漏れ回避

ここでは前節の課題を解決するための，VP 木構築時と検索時で異なる編集距離を使い分けるという解決策について述べる。

前項の課題のために，新しい編集距離計算ルールに基づく VP 木では枝刈りしてはいけないことになってしまい，このままでは VP 木を利用するメリットが失われる。そこで VP 木構築時には以下のルールを新たに設定する。

VP 木構築ルール(1)

数値選択メタ文字を含む文字列と含まない文字列とは分けて別々に VP 木を構築する

検索文字列は数値選択メタ文字を含まないので，このルールによって，数値選択メタ文字を含まない VP 木には通常の編集距離での計算と枝刈り条件が適用できる。

さらに，数値選択メタ文字列の VP 木については次のルールを設定する。

VP 木構築ルール(2)

数値選択メタ文字以外の数値部分をメタ文字記号で囲む

このルールによって，数値選択メタ文字列間の編集距離の計算時には編集距離ルール(3)ではなく編集距離ルール(4)が適用され，数値選択メタ文字列間の編集距離は Hausdorff 距離となる。その結果メタ文字を含まない文字列 q とメタ文字列 U, Vとの間の距離の三角不等式として

$$d(q,U) \leqq d(q,V) + d(V,U)$$

が成り立つようになるため，VP 木の左側の枝刈り前提条件となる三角不等式が成り立つようになる。したがって数値選択メタ文字列の VP 木でも左側の枝刈り条件が成り立つ場合には左側は枝刈りができる。

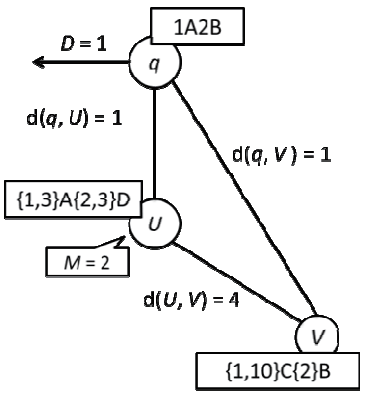

図 2　右側の枝刈り条件が成立しても枝刈りしてはいけない例

しかし右側は依然として枝刈りはできない。距離の三角不等式 $d(U, V) \leqq d(U, q) + d(q, V)$は一般には成り立たないからである。図 2 に例を示す。検索文字列 q（文字列:1A2B）と検索対象ノード U(文字列:{1,3}A{2,3}D, 中央値 M=2)，V（文字列：{1,10}C{2}B）について，ノード Uの右側の枝刈り条件が成り立っているが，ここで枝刈りを行うと抽出されるべき Vを逃してしまう。そこで VP 木検索時には以下のルールを設定する。

VP 木検索ルール

数値選択メタ文字列 VP 木の検索では，左側の枝刈り条件が成り立てば左側は枝刈りするが，右側については枝刈り条件が成立するか否かを問わず常に検索する

このルールによって VP 木全体の右側が枝刈りしてはいけないことになるのではなく，各ノードの右側を枝刈りしてはいけないだけである。

5．データによる「漏れなく高速化」の検証

ここでは，解決策をあるデータで検証した結果について述べる。

5.1. 検証の環境とデータ

数値選択メタ文字列240万を検索対象文字列として検証を行った。検索文字列となる誤入力は 5 万程度あるが，検証用に次のように誤入力 1 万を選んだ。

検索対象の数値選択メタ文字列240万の中からランダムに 15 万を選び，誤入力 5 万の検索を行った。検索のしきい値は誤入力の文字列長（数字文字列は単位文字扱い）の 25%（小数点以下切り捨て）とした。すなわち，誤入力の文字列長が 10 であればしきい値は 2 である。この検索で検索結果が 1 個以上あるものの中からランダムに 1 万件を選び，これを検証用検索文字列とした。検証の環境とデータについて表 1 にまとめた。

表 1　検証の環境とデータセット

検証用 PC	Dell Precision T5600 (RAM: 64GB)
上記の CPU	Intel Xeon E5-2650 2.0GHz
開発言語	Microsoft Visual C
検索対象データ	数値選択メタ文字列 240 万
検索文字列データ	誤入力（メタ文字なし）1 万

データ数の変化による性能の変化を見るために，データ数を次のように変化させた。上記 15 万件，その 15 万件を含む 30 万件，その 30 万件を含む 60 万件，その 60 万件を含む 120 万件, 全体 240 万件の 5 パターンである。

5.2. VP 木検索性能

VP 木構築時はすべてのノードで親ノードとなるノードはランダムに選ぶため，同じ誤入力の検索でも VP 木のどこにどの文字列があるかによって検索性能は異なる。そこで，誤入力 1 万件を 10 セットに分け，セットごとに（検索対象文字列は同じだが）VP 木を再作成して検索を行った。本来であれば 1 万の誤入力をグループ分けせず，この 1 万について異なる VP 木で検索を実施すべきであるが，検証に要する時間の都合でこのような設定を行った。乱数生成には Mersenne Twister[7]の C 実装を用いた。また，比較のために，VP 木を用いない場合の検索所要時間も測定した。いずれの場合も，検証に要する時間の関係で 1 回の測定しかしていない。

なお，VP 木を用いない場合の検索は全件検索を行う必要はない。文字列 s（文字列長を Ls とする）と文字列 t（文字列長を Lt とする）との間の編集距離 $d(s,t)$についてはその定義から $|Ls - Lt| \leqq d(s,t) \leqq Ls + Lt$ となる。これより $|Ls - d(s,t)| \leqq Lt \leqq Ls + d(s,t)$ が得られる。したがって検索対象文字列をその長さでグルーピングしておけば，文字列 q（文字列長を Lq とする）に対して編集距離 D以下の文字列を抽出する際には文字列長さ L が $|Lq - D| \leqq L \leqq Lq + D$ を満たす文字列についてだけ編集距離を計算すればよい。（以降，これを文字列長グループ検索と呼ぶ。）

図 3 に文字列長グループ検索と VP 木検索の検索所要時間のヒストグラムを示した。左列が文字列長グループ検索，右列が VP 木検索であり，上位行から下位行に向かって検索対象データ数が増えている。VP 木検索の検索所要時間はデータ数によらずおおむね文字列長グループ検索の 1/5 程度になっている。また，データ数が増えると文字列長グループ検索の場合はデータ数に比例する割合で検索所要時間が増加しているが，VP 木検索の場合の検索所要時間は線形よりも緩やかである。

また，今回の解決策を適用しない場合，VP 木検索でどの程度漏れが発生するのかを表 2 に示す。1 件の検索で 1 文字列でも漏れが発生した場合にはその検索は検索漏れとカウントし，データ数別に 1 万件の検索で何件漏れが発生したかを示す。なお括弧中のパーセンテージは，漏れ発生件数の検索 1 万件に対する比率である。

表 2 から，解決策を適用しない場合データ数が増えるにつれて漏れの発生頻度は増加することがわかる。

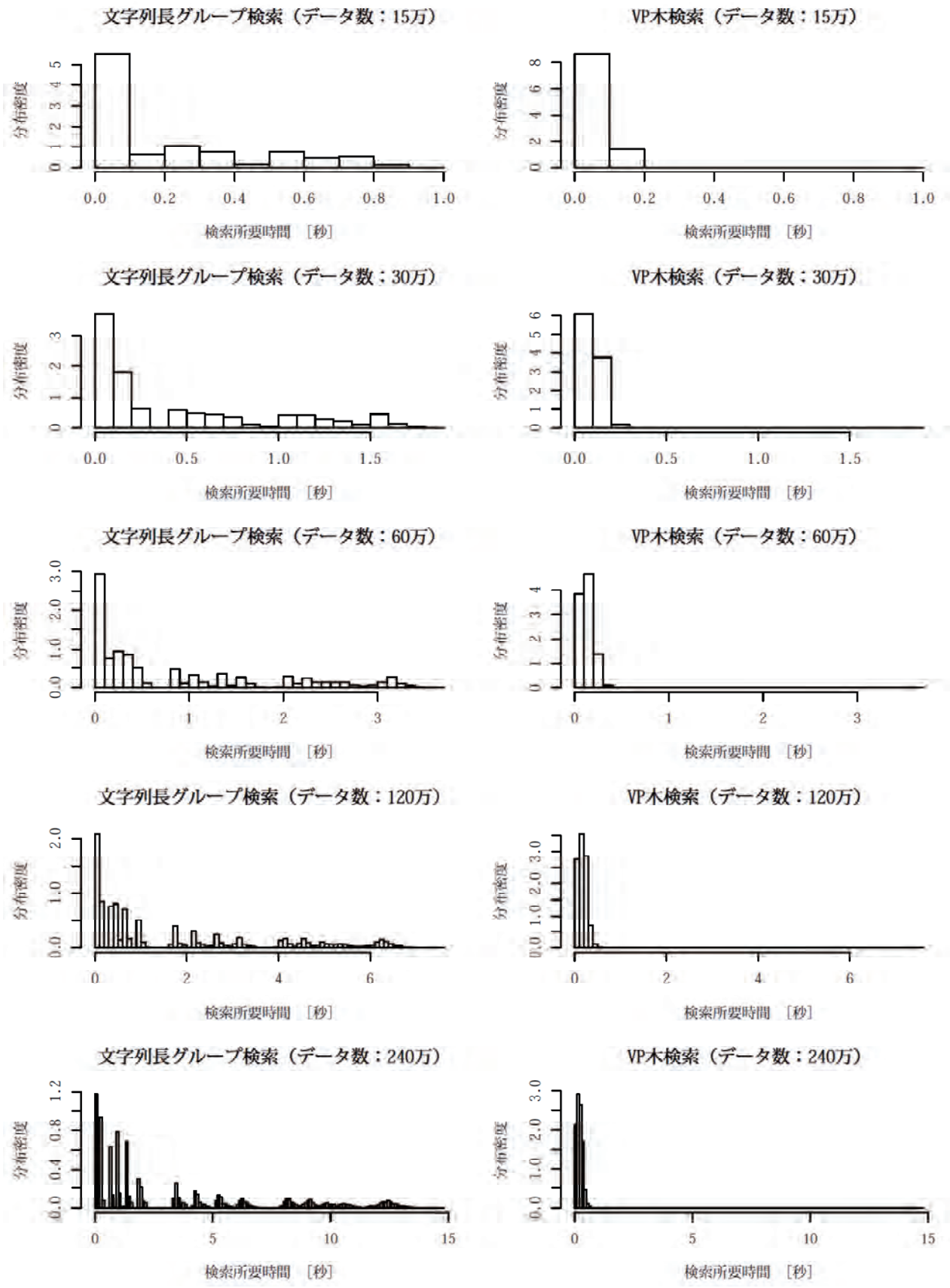

図 3　文字列長グループ検索と VP 木検索の性能比較

表 2　検索漏れの頻度

データ数	解決策非適用 （比率）	解決策適用 （比率）
15 万	982（10%）	0（0%）
30 万	1461（15%）	0（0%）
60 万	2073（21%）	0（0%）
120 万	2620（26%）	0（0%）
240 万	3193（32%）	0（0%）

5.3. VP 木生成性能

VP 生成に要する時間は，データ数を n とすると $O(n \log(n))$である [8]。図 4 に各データ数での 10 回の VP 木生成に要した時間の標本平均と標本標準偏差（エラーバー）を示す。破線はこのデータに対する $n \log(n)$のフィッティング結果である。

VP 木は検索のたびに作る必要はなく，VP 木を構成する文字列の変更がない限りは一度作った VP 木は使い続けることができる。図 4 からはデータ数が例えば 5000 万でも 5 時間かからないと期待されるため，データ数が多くてもデータの更新があった際に夜間バッチで VP 木を再構築できる。

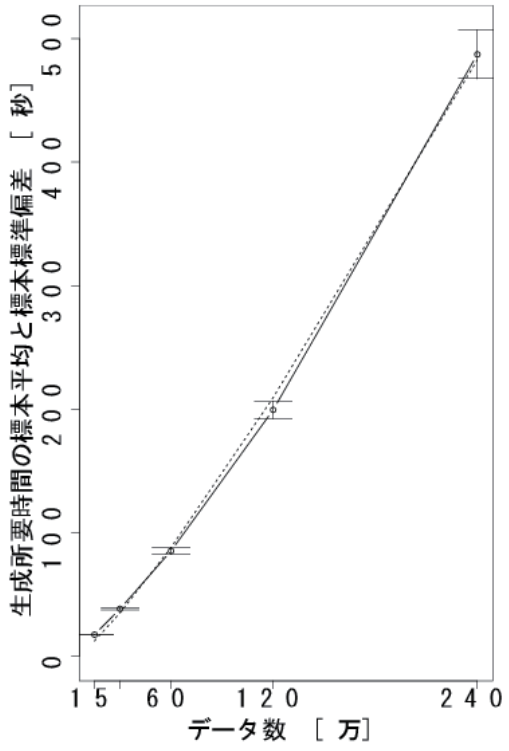

図 4　VP 木生成所要時間

6．おわりに

本報告では，数値選択メタ文字列に対する類似文字列検索を，VP 木を用いて高速に実施するための課題とその解決策そしてその検証結果について述べた。

これまでの研究で行った数値選択メタ文字列に対する編集距離の定義を用いて VP 木を構築して検索を行うと，検索漏れが発生するという課題があった。この課題に対して，VP 木の構築時は編集距離として Hausdorff 距離を用い，検索時はこれまでの研究で定義した数字文字列を単位文字とする編集距離を用いることで解決できることを示した。この解決策では，距離をインデックスとする木構造が持つ枝刈りのメリットが一部失われるが，それでもなお木構造を用いない検索よりも高速であることをデータで検証した。

今後は今回の解決策を，距離をインデックスとする VP 木以外のデータ構造にも適用できるようにし，解決策の応用範囲を拡大する。加えて，選択肢が数値以外の場合のメタ文字にも適用できるように拡張を行い，選択型のメタ文字を含む文字列に対する類似文字列検索技術の汎用化を図り，適用ビジネス領域を拡大していく。

参考文献

1) Myers，他：Approximate matching of regular expressions, Bulletin of Mathematical Biology, Vol.51, pp.7-37 (1989)

2) Muzatko：Approximate Regular Expression Matching, Proceedings of the Prague Stringology Club Workshop '96，pp.37-41 (1996)

3) Navarro：A guided tour to approximate string matching，ACM Computing Surveys，Vol.33，No.1，pp.31-88 (2001)

4) Wagner, 他：The string to string correction problem, Journal of the ACM，Vol.21，pp.168-178 (1974)

5) Chvez，他：Searching in metric spaces，ACM Computing Surveys，Vol.33，No.3，pp.273-321 (2001)

6) Hjaltason，他：Index-driven similarity search in metric spaces，ACM Transactions on Database Systems，Vol. 28，No.4，pp.517-580 (2003)

7) 松本，他：Mersenne Twister，http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html（2013 年 9 月 19 日アクセス）

8) Yianilos：Data structures and algorithms for

nearest neighbor search in general metric spaces, Proceedings of the Fourth ACM-SIAM Symposium on Discrete Algorithms, pp.311-321 (1993)

付録 A. 編集距離

編集距離は，2 つの文字列がどの程度異なっているかを定量的に表した指標のひとつである。文字列 s と文字列 t との間の編集距離（これを $d(s, t)$ と書く）は，s を構成する文字に対して以下の操作を施して t を生成する際の操作コストの和の最小値として定義される。

- 置換： s を構成する文字のひとつを，それとは異なる，t を構成するひとつの文字に変える操作
- 削除：s にあって t にない文字を s からひとつ削除する操作
- 挿入：s になくて t にある文字を s にひとつ挿入する操作
- 転置：s の 2 つの連続した文字並びを逆転させて，t の 2 つの連続した構成文字並びとする操作

これらの操作のコストを等しく 1 とする。これらの操作によって文字列 s から文字列 t を生成する方法はさまざまあり，その方法によって操作コストの和はさまざまな値をとる。編集距離は文字列 s から文字列 t を生成するその方法は問わず，上記操作コストの和の最小値として定義される。以下，文字列 solyusoin に対して上記文字操作によって文字列 solutions を生成する例を示す。

(1) solyusoin の 4 文字目の y の削除

solyusoin → solusoin

(2) 新しくできた solusoin の 5 文字目の s を t に置換

solusoin → solutoin

(3) 新しくできた solution の 6 文字目の o と 7 文字目の i とを転置

solutoin → solution

(4) 新しく出来た solution の 9 文字目（最後尾）に s を挿入

solution_ → solutions

なお，編集距離を計算する 2 つの文字列の長さは一般に異なってよい。

編集距離が大きいほど 2 つの文字列は異なっていることになるので，編集距離を用いて誤入力（検索文字列）と類似した文字列を検索対象文字列群から抽出することができる。検索前に編集距離のしきい値を決めておき，検索文字列に対する編集距離がそのしきい値以下の文字列を検索文字列と類似した文字列として抽出する。

なお，編集距離はその名にあるように距離の定義

- 距離が 0 となるのは 2 つの文字列が等しいときだけで，かつそのときに限られる：$d(s, t) = 0 \Leftrightarrow s$ と t は同一文字列
- 対称である：$d(s, t) = d(t, s)$ が成り立つ
- 距離の三角不等式が成り立つ：任意の文字列を u とするとき，$d(s, t) \leqq d(s, u) + d(u, t)$

を満たすことが知られている [4]。

文字操作コストとして置換・削除・挿入の 3 つを考慮する編集距離を Levenshtein 距離，これに加えて転置を考慮する編集距離を Damerau-Levenshtein 距離と呼ぶ。

付録 B. VP 木

VP 木は距離をインデックスとする 2 分探索木のひとつであり，以下の 3 点が子ノードを持つすべてのノードについて成り立つ。以降，子ノードと述べる際には孫以下のノードも含むものとする。

- ノードはそのノードの子ノードすべてとの間の距離の中央値 M を持つ
- ノードとそのノードの左側子ノードとの距離は，ノードが持つ中央値以下である
- ノードとそのノードの右側子ノードとの距離は，ノードが持つ中央値より大きい

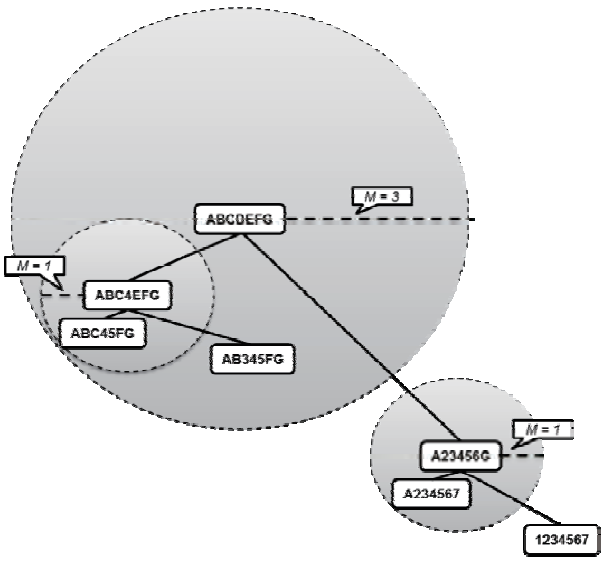

図 5　7 つの文字列からなる VP 木の例

図 5 に 7 つの文字列「ABCDEFG」「ABC4EFG」「ABC45FG」「AB345FG」「A23456G」「A234567」「1234567」からなる VP 木の例を示す。ノード

「ABCDEFG」とすべての 6 つの子ノード（編集距離の中央値 $M = 3$）について，また「ABC4EFG」とその 2 つの子ノード（編集距離の中央値 $M = 1$）について，そして「A23456G」とその 2 つの子ノード（編集距離の中央値 $M = 1$）について，それぞれ上記が成り立っている。

今，検索文字列を q，VP 木のあるノードの文字列を p，文字列 p のノードの任意の左側子ノード文字列を l，文字列 p のノードの任意の右側子ノード文字列を r とする。検索文字列との編集距離が D 以下の文字列を VP 木で検索する場合，次のことが言える。

(1) $D + M < \mathrm{d}(q, p)$ が成り立てば，そのノードの左側の枝は検索しなくてよい（以降，左側の枝刈りと呼ぶ）

$D + M < \mathrm{d}(q, p)$ が成り立つとき，距離の三角不等式 $\mathrm{d}(q, p) \leqq \mathrm{d}(q, l) + \mathrm{d}(l, p)$ から $D < \mathrm{d}(q, l)$ となる。これは文字列 p のノードの左側の任意のノード文字列について成り立つ。したがって $D + M < \mathrm{d}(q, p)$ が成り立つとき，文字列 p のノードの左側のノード文字列で検索文字列 q との間の編集距離が D 以下になるものは存在しない。図 6 に q，p，l の関係を示す。文字列 p のノードの左側すべての子ノードは文字列 p のノードから半径 M の内側にあるので，$D + M < \mathrm{d}(q, p)$ が成り立つときは文字列 p の左側のどの文字列も文字列 q からの（編集）距離は D よりも大きく（遠く）なる。

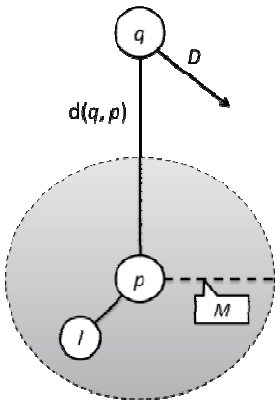

図 6　文字列 q，p，l の距離の関係

(2) $D + \mathrm{d}(q, p) \leqq M$ が成り立てば，そのノードの右側の枝は検索しなくてよい（以降，右側の枝刈りと呼ぶ）

$D + \mathrm{d}(q, p) \leqq M$ が成り立つとき，距離の三角不等式 $\mathrm{d}(p, r) \leqq \mathrm{d}(p, q) + \mathrm{d}(q, r)$ から $D < \mathrm{d}(q, r)$ となる。これは文字列 p のノードの右側の任意のノードの文字列について成り立つ。したがって $D + \mathrm{d}(q, p) \leqq M$ が成り立つとき，文字列 p のノードの右側ノード文字列で検索文字列 q との間の編集距離が D 以下になるものは存在しない。図 7 に q，p，r の関係を示す。文字列 p のノードの右側のすべての子ノードは文字列 p のノードから半径 M の外側にあるので，$D + \mathrm{d}(q, p) \leqq M$ が成り立つときは文字列 p の右側のどの文字列も文字列 q からの（編集）距離は D よりも大きく（遠く）なる。

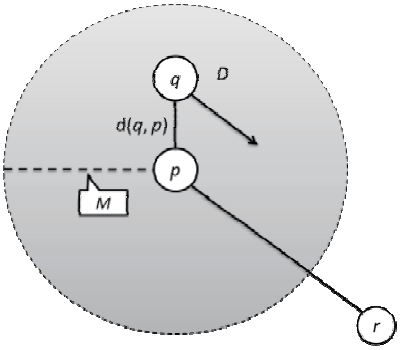

図 7　文字列 q，p，r の距離の関係

森川　浩司　　2001 年入社
事業企画開発本部　研究開発部
データ分析関連技術を核とした事業開発
koji.morikawa.hz@hitachi-solutions.com

高梨　勝敏　　1995 年入社
地域復興貢献室
地域復興貢献事業，コミュニティを主体とした知識交流システムの研究開発
katsutoshi..takanashi.ze@hitachi-solutions.com

宗形　聡　　2003 年入社
事業企画開発本部　研究開発部
統計・数理的アプローチによる業務分析，意思決定支援技術の研究開発
satoshi.munakata.tu@hitachi-solutions.com