解説

圧縮されたテキスト上のパターン照合
―データ圧縮とパターン照合の新展開―

竹田 正幸 takeda@i.kyushu-u.ac.jp　篠原 歩 ayumi@i.kyushu-u.ac.jp
九州大学大学院システム情報科学研究院情報理学部門／
科学技術振興事業団さきがけ研究 21

与えられたテキストの中から目的のパターンを見つけ出すという文字列パターン照合問題は，文字列処理に関する最も基本的な問題の１つである．近年では，さまざまな手法によって圧縮されたデータを，展開することなくそのままの形で文字列照合しようという試みもなされている．圧縮されたデータから目的のパターンを見つけ出すためには，通常ならばいったんハードディスク上にデータを展開し，それから既存の文字列照合アルゴリズムを走らせることになろう．しかし，この方法では，展開作業のために余分な時間と記憶領域が必要であり，たとえば遺伝子配列データのように，きわめて長い大量のデータを扱う場合には現実的ではない．そこで，テキストデータを圧縮したままパターン照合を行う「圧縮テキスト上でのパターン照合」が新しい研究課題として脚光を浴びるようになった．本稿では，この課題に関する最新の研究成果について，理論と実用の両面から解説する．

圧縮テキスト上のパターン照合とは

ネットワークを介してファイルをやりとりする際には，ファイルを圧縮してから相手に送ることが当たり前になってきました．このため，直接計算機関連の仕事に従事していない一般の方々にも「データ圧縮」はすっかりおなじみになったようです．画像データの圧縮では，損失のある圧縮（不可逆的な圧縮）がよく用いられます．画像の場合，見た目が変わらなければ，損失があってもあまり問題になりません．しかしテキストデータの場合には，元のデータが完全に復元できなければ困りますから，損失のない圧縮（可逆的な圧縮）が用いられます．この記事では，以後「データ圧縮」といえば損失のない圧縮を指すものとします．

さて，データ圧縮の目的といえば，

(1) ファイルサイズを小さくしてディスクを有効に使うこと，

(2) データ転送時の転送量を抑えること，

の２つが挙げられます．現在では，ノートパソコンですら，数十ギガバイトのハードディスクを搭載することが当たり前になりました．それでは，もうディスク容量が足らなくて困ることはないかというと，そうとばかりもいえません．というのも，ユーザは容量があればあったで詰め込める限りの情報を入れようとしがちだからです．

テキスト情報を対象とした検索システムにおいては，データが巨大になると圧縮して格納することが一般的です．また，電子メールやニュース記事，事務書類など，１つ１つは小さいファイルですが，チリも積もればなんとやらですので，古いものは圧縮しておくことが多い．これらのファイルは必要な時にはその都度展開して使うことになります．でも，肝心なときに，どこにどんなファイル名で置いていたか思い出せないことがあります．そんなときは，圧縮されたおびただしい数のファイル群から，あやふやな記憶を頼りに目的のファイルを探さなければなりません．

圧縮していない通常のテキストファイルであれば，Unix ユーザなら grep などの文字列サーチツールを使うところです．でも，圧縮してあるファイルについて文字列サーチをしたい場合，どうしたらよいでしょうか？

展開「してから」サーチ法．　もちろん答えは簡単で，

いったん展開してからサーチすればよろしい．つまり，

```
zcat file.Z | grep pattern
```

のように．シェルスクリプトを書いて1つのコマンドにしておけば便利でしょう（そのようなツールとしてzgrepが知られています）．

たしかに，これでいちいち展開する手間は省けるのですが，「展開に要する時間」＋「文字列サーチに要する時間」がかかるために，かなり遅い．

展開「しながら」サーチ法． もっと速くしたければ，次のような方法があります．展開プログラムの中に文字列サーチルーチンを組み込んで，コンパイルし直すのです．展開される端からサーチするように．ちょっと泥臭いけれども，これでかなり高速になります．

展開「しないで」サーチ法． さらにもっと速くサーチするにはどうしたらいいでしょうか？ 圧縮にcompressやgzip, lhaなどのツールを用いる場合，「展開時間」と「サーチ時間」を比べてみると，圧倒的に展開に時間を喰っていることが分かります．それでは，展開せずに直接サーチすることはできないでしょうか？ もしそれができれば，「してから／しながら」法より高速にサーチできるかもしれません．「圧縮テキスト上のパターン照合」というテーマは，こうして生まれました（図-1参照）．

「圧縮テキスト上のパターン照合」と書くと長いので，以後は，圧縮パターン照合（compressed pattern matching）と呼ぶことにします．圧縮パターン照合問題を定式化したのは，1992年のAmirとBensonの論文が最初とされています．この論文では，2次元配列データをrun-length符号によって圧縮したものを展開せずにサーチする問題を扱っています．これに続いて，1994年，Amir, Benson, Farachによって LZW法，1995年，FarachとThorupによってLZ77法に対する圧縮パターン照合アルゴリズムが開発されました[☆1]．しかし，これらの研究はもっぱら理論的興味からなされたもので，「展開しながら／してから」サーチするより実際に高速かどうかは問題にされていませんでした．Amirらは，圧縮パターン照合アルゴリズムは，圧縮テキストのサイズnに比例した時間でサーチを行える場合に効率的である，と定義していますが，その根拠は，もとのテキスト長Nに対して，圧縮テキスト長nが極端な場合には$\sqrt{N}$や$\log N$に比例することによるものです．しかし，Amirらも認める通り，実用的見地からは，nはNに比例しており，その比例常数こそが問題となります．最近FarachとThorupのアルゴリズムを実働化したC++プログラムソースを入手したので実行してみたのですが予想通りとてつもなく遅く，プログラミング技術の問題はあるにせよ，「展開してから／しながら」サーチする方が格段に速いことが実証されました．

Amirらのアルゴリズムについても，その実用性にはやはり疑問が投げかけられていました（Manber 1994など）．しかし，筆者らの研究グループは，このアルゴリズムに着目し，複数パターンへの拡張とともに，効率的な実装法を探り，「してから」法や「しながら」法に比べ2倍程度高速であることを示しました（Kida ほか 1998）．

図-1 展開してから／しながら／しないでサーチ[☆2]

筆者らが小学生の頃，「ウルトラマンと仮面ライダーがインスタントラーメンの早喰い競争をしました．どっちが勝ったでしょう？」といったクイズが流行りました．答えは「ウルトラマンはラーメンができるまでの3分間が待てないから仮面ライダーの勝ち」です．お湯をかけてできあがるのを待っていたのでは仮面ライダーに勝てません．勝つためには，お湯をかけて麺がほぐれるはしから食べていくか（展開しながら法），お湯をかけずにいきなり食べるか（展開しないで法）しかありません．しかし，単に早喰いを競うのではなく，我々は文字列サーチをしなければならないのでした．丸飲みして何を食べたか分からないようでは困ります．では，高速に味わうにはどうしたらよいでしょうか？ 続きをお読みください．

☆1 LZ77法は，gzip, pkzip, lhaなどの圧縮ツールで用いられている圧縮法であり，LZW法は，Unixのcompressコマンドなどで用いられています．その詳細については，文献3）などをご覧ください．

☆2 筆者らがもともと用意していた絵は，円谷プロにキャラクタ使用料を支払うという条件の下でいったんは掲載を許可されましたが，その後「キャラクタイメージを損なう」との理由で掲載不許可となり，やむなくこのような絵に差し替えました．できましたらキャラクタを頭の中で適当に置き換えながらお読みください．

これは，圧縮パターン照合アルゴリズムの世界で最初の実働化であり，当該分野の研究の実用的価値を最初に示したものと評価されています．また，ビットパラレリズム（bit-parallelism）と呼ばれる技法を応用することでさらに高速化することに成功しました（Kida ほか 1999）．

従来，データ圧縮法の評価基準は，(1) 圧縮率がどのくらい良いか，(2) 圧縮・展開に要する時間がどのくらいか，の 2 点でした．Amir らは，ここに新しい評価基準，すなわち「圧縮したままの文字列照合がどのくらい高速に行えるか」という基準を持ち込みました．もちろん，従来の評価基準を否定するものではありません．が，ディスクの大容量化・低価格化が進み，CPU パワーが劇的に向上した今日においては，「圧縮率は多少犠牲にしてもよいし，圧縮に多少時間がかかっても 1 回きりなら我慢できる．しかし，サーチや（部分的な）展開はたびたび行うので，できるだけ時間を短縮したい」というケースは，少なくありません．LZ77 圧縮法は圧縮率が高いことと展開時間が比較的短いことにより，gzip をはじめ多くの圧縮ツールで使用されていますが，圧縮パターン照合の観点からは必ずしも良いものとはいえません．

この分野では，既存の圧縮技法に対して高速な圧縮パターン照合アルゴリズムを考案することにとどまらず，これまで省みられなかった圧縮法を再評価したり，新しい評価基準のもとで有効な新しい圧縮技法を開発することが重要です．そのためには，種々の圧縮法を統一的に扱う理論的枠組みを用意し，そのもとで圧縮パターン照合に関する議論を展開することが必要です．そこで，筆者らは圧縮テキストをコラージュシステム（collage system）と名付けた形式的体系に抽象化し，コラージュシステム（として表現されたテキスト）を対象として圧縮パターン照合問題を考えることにしました．コラージュシステムは，LZ77 法や LZ78/LZW 法などの古典的辞書式圧縮法はもちろんのこと，比較的最近の研究である Witten らの Sequitur，Larrson らの Re-pair，Kieffer らによる「文法変換に基づく圧縮法」もカバーしています．つまり，個々の圧縮法ごとにパターン照合アルゴリズムを開発するのではなく，コラージュシステムの枠組みのもとで一般的な議論を展開してゆくことができるのです．

パターン照合アルゴリズムとテキスト圧縮法

文字列パターン照合問題とは？

文字列パターン照合問題とは，パターンとテキストという 2 つの文字列が与えられたときに，テキスト中におけるパターンの出現位置をすべて求めよ，という問題です．この問題に関する研究は，1974 年の Knuth, Morris, Pratt らによる KMP アルゴリズムの出現以来，古くから盛んに行われ，現在もなお活発に研究が行われています．主なものとして，(1) Knuth-Morris-Pratt (KMP) 法，(2) Shift-Or 法，(3) Boyer-Moore (BM) 法が知られています．アルゴリズムの詳細にふれる余裕はないのですが，後の説明に必要となりますので，KMP アルゴリズムについてだけ簡単に説明しておきましょう．図 -2 に，パターン *abacb* に対する KMP オートマトンを示しました．また，このオートマトンをテキスト *abacbbabaabacb* 上を走査させた際の状態遷移も示しています．

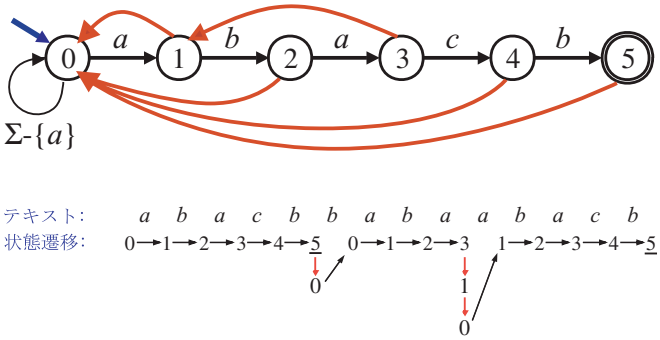

図 -2　パターン *abacb* に対する KMP オートマトン（上）とテキスト *abacbbabaabacb* に対するその動作（下）

上段の図において，円は状態を表し，黒い矢印は goto 関数，赤い矢印は failure 関数，2 重円は受理状態をそれぞれ表す．また，下段の図において，黒い矢印と赤い矢印は，それぞれ，goto 関数，failure 関数による状態遷移を表す．

KMP オートマトンを高速に動作させるためには，failure 遷移（図の赤い矢印で示した遷移）を除去して決定性のオートマトンに変換した状態遷移表を，状態数×256 の 2 次元の配列に格納し，テキストの文字（1 バイト）ごとにこの表を参照して状態遷移を続けます．KMP 法には，日本語テキストをうまく扱うことできるという利点があります．どういうことかというと，日本語テキストには 1 バイト文字と 2 バイト文字が混在するため，素朴なアイディアとしては，1 バイト文字に詰め物をして 2 バイトにしてから照合する方法がありますが，これでは速度が低下してしまいます．ところが，KMP オートマトンにちょっと手を加えるだけで，1 バイト単位にオートマトンを走らせながら，符号語のずれ読みなく照合を行うことができるのです．このテクニックは，Shift-Or 法や BM 法には適用できません．詳しくは文献 1）を参照してください．

テキスト圧縮法

テキスト圧縮技法について簡単に説明しておく必要が

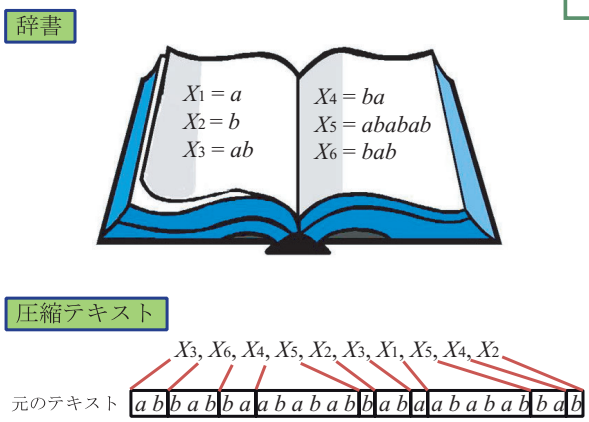

図 -3　辞書式圧縮法

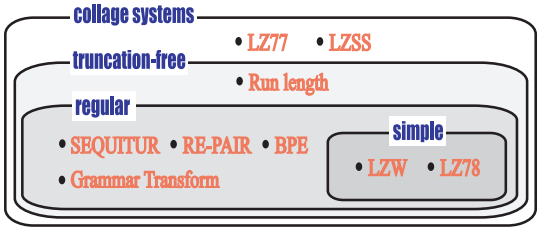

図 -4　コラージュシステムの階層

ありますが，筆者らはデータ圧縮の専門家ではありませんので，ここでは，以下の議論に必要最小限にとどめたいと思います．興味のある方は，本会誌第 42 巻第 1 号に群馬大の横尾英俊先生による解説記事が載っていますので，ぜひそちらをご覧ください．

さて，gzip, lha などのツールに用いられている LZ77 圧縮法や，compress で用いられている LZW 法などは，辞書式圧縮法と呼ばれる範疇に分類されます．辞書式圧縮法とはどのような圧縮法を指すのでしょうか？　ここで図 -3 をご覧ください．辞書には文字列が登録されており，各文字列にはシリアルナンバー（トークン）が付いています．テキストの圧縮は，まず (1) テキストを辞書中の文字列に分解し，次に (2) 各文字列を対応するトークンで置き換える，という手順で行われます．その結果，テキストはトークンの列に変わります．

辞書の具体的な作成の仕方，(1) の分解の仕方，(2) で得られるトークン列の具体的な符号化の方法こそが各々の圧縮法のウリには違いないのですが，本稿はあくまでパターン照合の観点からみていますので，それは捨象してしまいます．ここで強調しておきたいのは「圧縮テキストはトークンの列である」ということです．もちろん，辞書がないと復元ができませんから，結局「圧縮テキストは辞書の符号化とトークン列の符号化の 2 つから成る」ことになります．

圧縮テキスト上のパターン照合：理論編

コラージュシステム

筆者らの研究グループは，圧縮パターン照合の観点から，コラージュシステム（collage system）と名付けた形式的体系によって，辞書式圧縮法による圧縮テキストを記述することを提案しました (Kida ほか 1999)．コラージュとは，ご存知のように，いろんな素材を切り貼りして作るものです．つまり，「文字列の切り貼り」操作に基づく文字列表現の形式体系，というつもりです．具体的な「切り貼り」操作としては，(1) 文字列の連接 (concatenation)，(2) 文字列の繰り返し (repetition)，(3) 文字列の先頭（末尾）の切り取り (truncation)，という 3 種類の操作を許しています．コラージュシステムは，辞書 D とトークン列 S から成ります．辞書は以下のような代入文の集まりです．

$$X_1 = a;$$
$$X_2 = b;$$
$$X_3 = X_1 \cdot X_2;$$
$$X_4 = X_2 \cdot X_1;$$
$$X_5 = (X_3)^3;$$
$$X_6 = {}^{[3]}(X_5).$$

ここで，$X_5 = (X_3)^3$ は，X_3 の表す文字列を 3 回繰り返したものを X_5 で表すことを意味します．また，$X_6 = {}^{[3]}(X_5)$ は，X_5 の表す文字列（*ababab*）の先頭 3 文字を切り落としたものを X_6 で表すことを意味しています．一方，S は D で定義されたトークンを

$$S = X_3, X_6, X_4, X_5, X_2, X_3, X_1, X_5, X_4, X_2$$

のように並べたものです．

主要な辞書式圧縮法による圧縮テキストは，コラージュシステムとして記述することができます．図 -4 に，各「切り貼り」操作の有無に基づくコラージュシステムの階層を示しました．また，その中にいくつかの圧縮技法を，その出力する圧縮テキストの属する階層に応じて示しておきました．

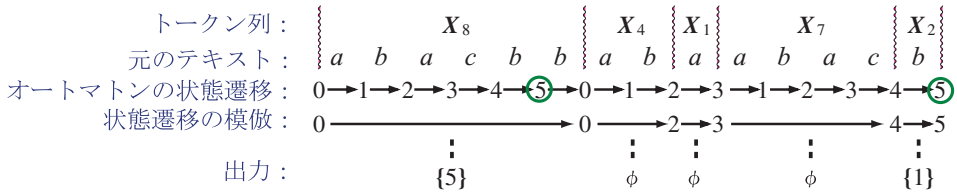

図 -5　コラージュシステム上のパターン照合

コラージュシステム上のパターン照合

筆者らは，テキストがコラージュシステムの形式で与えられた際に，その上で KMP アルゴリズムの動作を模倣するアルゴリズムを開発しました（Kida ほか 1999）．その詳細に立ち入る余裕はありませんが，図 -5 にその模倣の様子を示しました．このアルゴリズムの計算量についての理論的な結果は，次のようになります．

定理．コラージュシステムのパターン照合問題は，$O((\|D\|+|S|)\cdot height(D)+m^2+r)$ 時間と $O(\|D\|+m^2)$ 領域で解くことができる．また，切り取り操作を含まないコラージュシステムに対しては，$O(\|D\|+|S|+m^2+r)$ 時間で解くことができる．ここで，$\|D\|$, $|S|$ は，コラージュシステム $\langle D, S\rangle$ の辞書 D のサイズ，トークン列 S のサイズをそれぞれ表し，$height(D)$ は辞書 D の統語木の高さ，m はパターン長，r はテキスト中におけるパターンの出現回数である．

大雑把にいえば，$\|D\|+|S|$ が圧縮テキストのサイズに対応します．したがって，切り取り操作を含まない場合には，パターンの前処理の後，圧縮テキスト長に比例した時間でパターン照合が行えることになります．

図 -4 に示したように，gzip や lha 等のツールで用いられている LZ77 法によって圧縮されたテキストは，コラージュシステムとしてみると切り取り操作を含みますので，このアルゴリズムでは，圧縮テキスト長の線形時間では照合できません．また，はじめに述べたように，LZ77 に対するまったく別のアプローチによる Farach-Thorup 法も非常に遅いことが確認できましたし，ビットパラレリズムによる照合法を試した Navarro と Raffinot (1999) の実験においても，遅いという結果が出ています．つまり，LZ77 法は，圧縮テキスト上のパターン照合の観点からは，良い圧縮法ではないといえそうです．一方，LZW 法による圧縮テキストは，切り取り操作も繰り返し操作も含みません．したがって，上の定理から，圧縮テキスト長に比例した時間でサーチを終えることができることが分かり，高速な処理が期待できます．

筆者らはまた，KMP アルゴリズムだけでなく，BM アルゴリズムの動作を模倣するアルゴリズムの開発にも成功しています（Shibata ほか 2000）．

圧縮テキスト上のパターン照合：実用編

LZW 法に基づいた圧縮ツールである Compress を用いて，「してから／しながら」法と「しないで」法についての処理時間の比較を行いました．実験には 2 つのテキストを用いました．1 つは，代表的な DNA 塩基配列データベースである GenBank からファイルを 1 個選び塩基配列部分と基本的情報だけに絞ったもので，大部分が c,a,t,g の 4 文字から成っています．もう 1 つは，医学生物学関連の文献データベースである Medline の一部分で，大部分が英文要旨から成っています．図 -6 に実験の結果を示しました．比較のため LZ77 法に基づく gzip についての結果も示しています．この実験結果から，LZW 法の場合には圧縮テキストを展開しながら照合するよりも高速に照合できていることが分かります．

もっと野心的な目標：テキスト圧縮によるパターン照合の高速化!?

これまで述べた通り，LZW 法については，圧縮パターン照合により，「展開してから／しないで」照合するよりも高速にサーチを行うことができます．すなわち，

目標 I：

展開時間＋非圧縮パターン照合時間＞圧縮パターン照合時間

が達成できました．しかし，この圧縮パターン照合は，通常の非圧縮テキスト上のパターン照合時間と比べて 2 ～ 3 倍程度の時間を費やしています．つまり，圧縮の代償としてそれだけの時間を受け入れなければならないということです．もし，ディスク容量に十分余裕があり，

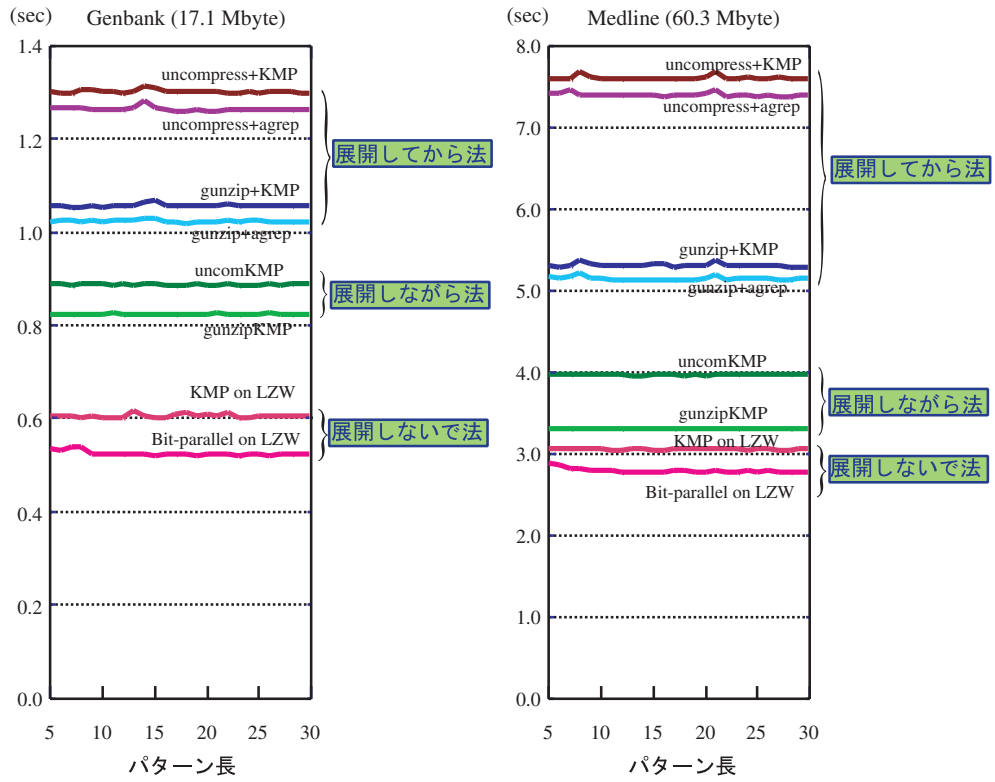

図 -6　実験結果：目標 I の達成

データ転送コストも気にならない状況だとすれば，圧縮などはしないでしょう．でももしそうなったら，圧縮パターン照合の研究など何の意味もないのでは？

実は，圧縮パターン照合には，まったく別の，もっと野心的な目標がありました．それは「テキスト圧縮によるパターン照合の高速化」です．つまり，テキストをあらかじめ圧縮しておくことにより，サーチ時間を短縮してやろう，という目論見です．ですから，「非圧縮テキスト上のサーチ」と比べて高速にサーチを行うことが目的となります．つまり，

目標 II：非圧縮パターン照合時間＋圧縮パターン照合時間

を狙うわけです．目標 I と比べると左辺から「展開時間」がなくなった分，ハードルは高くなっています．もちろん，データ圧縮によってデータ転送量を抑えれば，I/O 時間を短縮できますから，サーチに要するトータルの時間は短縮できそうです．しかし，CPU 時間は，非圧縮テキストに対するサーチと比べてかなり増大することが予想できます．したがって，データ転送コストが比較的小さい場合やディスクキャッシュが効いている状況では，必ずしも高速化されません．

初めにふれた Amir らの先駆的研究には目標 II のような狙いはありませんでした．筆者らがこのアイディアを初めて耳にしたのは，Amir らの研究からさらに数年遡った 1989 年の暮れ，現九州工業大学情報工学部教授の篠原武先生からでした．そのアイディアとは，ハフマン符号によって圧縮したテキストをそのままパターン照合する Aho-Corasick パターン照合機械（KMP オートマトンの複数版）についてでした．初期の研究成果は，1992 年の 1 月に「情報学シンポジウム」で発表され（深町ほか「可変長符号圧縮データのための文字列パターン照合ーゲノム情報の高速検索技法」），さらに進んだ結果が，文献 2）で報告されています．

圧縮による高速化では，照合時間を圧縮率と同程度に短縮することが 1 つの目標となります．たとえば，圧縮ファイルのサイズがもとのファイルの 50% だとすると，いかにして照合時間を非圧縮ファイルの場合の 50% 程度にまで短縮するかが課題です．文字単位のハフマン符号は，目標であるこの圧縮率の値があまり良くありません．では他のもっと圧縮率の良い圧縮法を選べばいいのかというと，今度は圧縮パターン照合を高速に行えるとは限りませんから，そう簡単にはいきません．筆者らは，バイト対符号化（Byte Pair Encoding; 以下 BPE と略記）と呼ばれる圧縮法に着目しました．この圧縮法は，圧縮率，

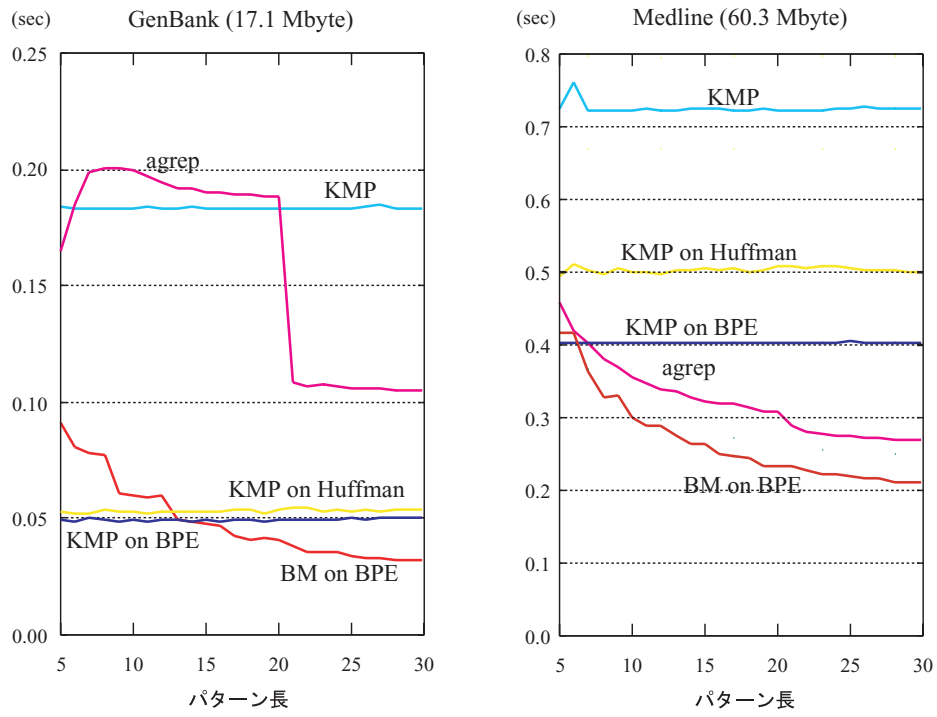

図 -7　実験結果：目標 II の達成

圧縮時間といった古典的な観点からは必ずしも優れた圧縮方法とはいえませんが，高速に圧縮パターン照合が行えるという意味で魅力的です．

筆者らの研究グループでは，コラージュシステムに対して開発した KMP 型および BM 型の 2 つの汎用パターン照合アルゴリズムをこの BPE 法に特化して実装し，実験を行いました．その結果，最も高速な文字列照合ツールとして知られる Agrep と比べて 1.2 ～ 3 倍高速となることが判明しました（図 -7 参照）．こうして，より大胆な目標である「テキスト圧縮によるパターン照合の高速化」をも達成できることが示されました．

まとめ・今後の展開

圧縮テキスト上のパターン照合問題は，もともとは理論的な興味からの研究が中心でしたが，研究が深まるにしたがって，むしろ実用的な観点からもきわめて重要な技術になってきたといえます．特に，目標 I のみならず，目標 II が達成できたことは意義深いものといえるでしょう．つまり，これまではディスク容量節減のために「消極的に」圧縮を行っていたユーザが，ディスク容量が十分ある場合でも，パターン照合の高速化を目的にして，むしろ「積極的に」圧縮を行うことになるかもしれません．

本稿では，単一の文字列パターンが部分列としてテキストに現れる位置を求める問題のみを扱いましたが，(1) 複数文字列照合，(2) 近似文字列照合，(3) 正規表現の照合，などを圧縮テキスト上で行う研究が，筆者らのグループ，チリ大学，フィンランドのヘルシンキ大学のグループなどの手によって推し進められています．こうしてテキストを対象としたさまざまな処理を圧縮したままで高速に行えるようになれば，テキストファイルは圧縮しておいておくのが常識だ，という状況が実現するかもしれません．

なお，圧縮パターン照合に関する技術的な詳細については，文献 4）およびそこで引用した参考文献などをご覧ください．

参考文献
1) 有川節夫 , 篠原　武 : 文字列パターン照合アルゴリズム , コンピュータソフトウェア , Vol.4, No.2, pp.2-23 (1987).
2) 宮崎正路 , 深町修一 , 竹田正幸 , 篠原　武 : 圧縮テキストに対するパターン照合機械の高速化 , 情報処理学会論文誌 , Vol.39, No.9, pp.2638-2648 (Sep. 1998).
3) Nelson, M. and Gailly, J.-L: The Data Compression Book, 2nd Edition, M & T Book (1995).
邦訳 : 萩原剛志 , 山口　英 訳 : データ圧縮ハンドブック－マルチメディアデータ圧縮の実践的プログラミング技法 , プレンティスホール出版 , トッパン (1996).
4)　Takeda, M. et al.: Speeding up String Pattern Matching by Text Compression: The Dawn of a New Era, 情報処理学会論文誌 , Vol.42, No.3 (40 周年記念論文特集号), pp.370-384 (Mar. 2001).

（平成 14 年 3 月 29 日受付）