算術符号を用いたウェブグラフ表現のための圧縮方法

シュウテイ† 片山 薫†

† 首都大学東京システムデザイン研究科 〒191–0065 東京都日野市旭が丘 6-6
E-mail: †shuu-tei@ed.tmu.ac.jp, ††kaoru@tmu.ac.jp

あらまし ウェブグラフとは，クローリングされたウェブページをノード，ウェブページ間のハイパーリンクをエッジとして構築されるグラフである．近年，インターネットの急速な発展に伴い，大規模なウェブグラフを効率的に保存することはますます重要になっている．ウェブグラフを圧縮する際，圧縮するとともにウェブグラフの情報を効率的に検索することも求められる．本稿では，算術符号を用いたウェブグラフの圧縮方法を提案する．ベンチマークデータを用いて，Boldi と Vigna が提案した WebGraph 等の既存の方法と比較した．

キーワード 算術符号，ウェブグラフ，圧縮

1. はじめに

ウェブグラフは極めて大規模になっており，その解析は年々困難になっている．例えば，WorldWideWebSize.com (http://www.worldwidewebsize.com/) によると，2012 年の Google のインデックスについて，500 億ぐらいのウェブページがあると推定されている．この事実を背景に，さまざまな手法が研究されている．Boldi と Vigna [1] [9] はウェブグラフのページに自然数を付け，ウェブページの隣接リストを作成し，この隣接リストの形式を変換し，圧縮するという WebGraph 手法を提案した．また，Claude と Navarro [2] はすべての隣接リストを結び付け，辞書式の圧縮手法でリストを圧縮するという Re-pair 手法を提案した．一方，Grabowski と Bieniecki [3] が提案した SSL 手法は Boldi と Vigna の手法に似ている．しかし，Grabowski と Bieniecki は圧縮率を上げるため，アクセス時間を犠牲にし，よりコンパクトな手法を提案した．

本稿では，Boldi と Vigna が提案した *gap* [1] 手法と算術符号を組み合わせた圧縮手法を提案する．圧縮する前にすべてのウェブページに自然数を付け，ページの隣接リストを作成し，順番に整列する．*gap* とはウェブグラフの以下に述べる性質を利用した手法である．順番に整列したページの隣接リスト上の数字は連続する傾向がある．なぜなら，階層のあるウェブページでは，メインページの階層が一番高く，含まれているページは順序が後ろのページほど階層が次第に低くなる．階層の下部にあるページ内のリンクは、辞書式順序で隣接している傾向がある．ほとんどのページは、階層の固定レベルを指しているリンクがある。また，階層が高い位置のウェブグラフページがほとんどのページで指されている．したがって，一つのページに含まれる隣接ページらの階層位置が近い，隣接リスト上の数字が連続値なりやすい．そこで，一つ隣接リスト上の連続した自然数間の差の絶対値をそれぞれ計算すればリストに現れるすべての数字が小さくなる．算術符号を用い，このページの差の絶対値を計算したリストを圧縮する．

本稿は以下の内容で構成される．第二章ではウェブグラフに関する関連研究について言及する．第三章では本稿に関する予備知識を述べる．第四章では提案手法を紹介する．第五章では提案手法の実験結果と実験評価について述べる．第六章ではまとめと展望を述べる．

2. 関連研究

今実際にウェブグラフを利用した解析手法には主にランキングアルゴリズムとコミュニティ抽出手法がある．Google の PageRank [10] や，Microsoft の SALSA [11] や，trawling [12] などがある．一方，ウェブグラフの圧縮方法について基本的に二種類に分かれる．一つはウェブグラフを複雑な構造から簡単な構造に変換するという方法である．もう一つはウェブページを符号化し，インデックスを作成し，圧縮するという方法である．

ウェブグラフの構成改良方法は一般的にウェブグラフのノード数とエッジ数を削減することを目指している．この方法には Buehrer らが提案した Virtual Node [4] や，Navalakha らが提案した graph summary の圧縮手法 [5] などがある．Buehrer らは Virtual Node というウェブグラフのノードを増加しながら，エッジを減少させる手法を提案した．Virtual Node [4] は Min-Wise Independent Permutations [13] という置換族によるハッシュ法を利用し，リンクを置き換え Virtual Node と呼ばれる新しいノードを作成するという手法である．Navalakha らが提案した graph summary の各ノードは，もとのグラフノードのセットに対応する集合グラフであり，各エッジは，2 つのセット内のノードのすべての対応するエッジを表すという手法である．

符号化を用いる圧縮方法には，Boldi と Vigna が提案し

た WebGraph や，Claude と Navarro が提案した Re-Pair や，Grabowski と Bieniecki が提案した SSL などがある．これらの方法として代表的な方法は WebGraph である．WebGraph とはウェブページに自然数を付け，隣接リストを効率よく圧縮する手法である．彼らは *MG4J*（Managing Gigabytes forJava：ファイルのインデックスを作成する Java のプログラム）ですべてのウェブページに自然数を付け，ページの隣接リストを作成する．このとき，一般的に任意な二つの隣接リストは共通点があまりない，あるいは，大部分の部分数字列を共有するという特徴がある．そこで，参照ページの隣接リストを指定し，*copylists* という数字リストを作成する．*copylists* をもとの隣接リストの代わりに圧縮する．隣接リストが省略でき，圧縮する数字も簡単になる．参照ページ以外のページの隣接リストを参照ページの隣接リストと比較する．ある数字が参照ページの隣接リストにあり，自分の隣接リストにない場合は，*copylists* に “0” として保存する．参照ページの隣接リストにあり，自分の隣接リストにもある場合は， *copylists* に “1” として保存する．ある数字が参照ページの隣接リストになく，自分の隣接リストにある場合は，そのまま保存する．この手法の通りように，参照ページ以外のページの隣接リストは対応する *copylists* と残る数字で表すということですべてのウェブグラフを表現する．

［例 1］ 表 1 に示すように，ウェブページの隣接構造が三つの部分 *Node*，*Outdegree*，*Successors* で構成される [1]．*Node* とはウェブページのインデックスであり，*Outdegree* とはこのウェブページの隣接ノードの個数であり，*Successors* とはウェブページの隣接ノードのインデックス，つまり，このウェブページの隣接リストである．

*Node*7 の *Successors* を *Node*8 の *Successors* と比較すると，大部分の部分数字列を共有するという特徴が分かる．*Node*7 の *Successors* は *Node*8 の *Successors* の参考リストとする．そこで，*Node*8 の *Successors* 上の数字を *Node*7 の *Successors* 上の数字と順番に比較し，表 2 のように，*copylists* という “0” と “1” で構成される数字リストを作成する．*Node*7 の *Successors* の一番の数字は “8” であり，*Node*8 の *Successors* にないため，*Node*8 の *copylists* の一番の数字は “0” である．*Node*7 の *Successors* の二番の数字は “10” であり，*Node*8 の *Successors* にもあるため，*Node*8 の *copylists* の二番の数字は “1” である．*Node*7 の *Successors* の三番の数字は “11” であり，*Node*8 の *Successors* にもあるため，*Node*8 の *copylists* の三番の数字は “1” である．以上のステップを繰り返し，参考リスト，つまり，*Node*7 の *Successors* 上すべての数字を *Node*8 の *Successors* 上の数字と比較した後，*Node*8 の *Successors* に四つの数字 “17”，“311”，“322”，“355” が *Node*7 の *Successors* にないことが分かる，これらの数字を残る数字と呼び，*Node*8 の $Extra-nodes$ という数字リストに保存する．

*Node*10 の *copylists* と $Extra-nodes$ も以上のステップのとおり，作成される．

ほかの符号化を用いる圧縮方法について，Claude と Navarro が提案した Re-Pair はウェブグラフに表示され規則性を利用し，アクセス時間が速い手法である．Claude と Navarro はすべての隣接リストを結び付け，辞書式の圧縮手法 LZ78 でリストを圧縮するという方法を提案した．Grabowski と Bieniecki が提案した手法 SSL は Boldi と Vigna の手法に似ている．しかし，Boldi と Vigna の手法よりコンパクトな手法である．ます，隣接リストを作り，参照ページの隣接リストを指定し，WebGraph の *copylists* に似ている “0”，“1”，“2” と “3” で構成される $flag$ [3] という配列を作成する．参照ページの隣接リストと比較し，ある数字 j が参照ページの隣接リストにあり，自分の隣接リストもある場合は，$flag$ に “0” として保存する．また，このとき $j+1$ が自分の隣接リストにあるばあいは，$flag$ に “2” として保存する．$j+2$ が自分の隣接リストにある場合は，$flag$ に “3” として保存する．“0”，“2” と “3” で標識されていない場合は，$flag$ に “1” として保存する．残る数字を $Extra-nodes$ という数字リストに保存する．

表 1　ウェブページの隣接構造.

Node	*Outdegree*	*Successors*
...	...	...
7	11	8,10,11,12,13,14, 18,19,198,310,2024
8	10	10,11,12,17,18,19,310,311,322,355
9	0	
10	5	8,10,11,12,80
...	...	...

表 2　*copylists* の表現形式.

Node	*Outdegree*	*copylists*	$Extra-nodes$
...	...	...	...
7	11		8,10,11,12,13,14, 18,19, 198,310,2024
8	10	01110011010	17,311,322,355
9	0		
10	5	11110000000	67
...	...	...	...

3. 準　　備

3.1 ウェブグラフ

ウェブグラフとは，ウェブページをノード，二つのウェブページの間のハイパーリンクをエッジとする有向グラフである．つまり，ウェブグラフを $G=(V,E)$ として表せば，V はウェブグラフ中のノード集合であり，E はウェブグラフ中のエッジ集合である．そこで，$G=(V,E)$ においてグラフ中に含まれるノード数を $|V|$，エッジ数を $|E|$ と表す．すべてのウェブページに自然数を付け，一つのノードの隣接リスト上の数字は隣接ノードのインデックスである．各隣接リストに含まれるインデックスの総和は $|E|$ と等しい．ウェブグラフには三つの特性がある [1].

- 局所性

一つのウェブページの隣接リスト上の数字はそのウェブページの隣接ページのインデックスを表す．これらの隣接ページの URL 文字列を比較すれば，部分ずつ URL 文字列の先頭文字列が共有されることが分かる．つまり，URL 文字列の先頭文字

表 3　*gap* でウェブページの隣接構造.

Node	*Outdegree*	*Successors*
...	...	...
7	11	2,1,0,0,0,0,3,0,178,111,1713
8	10	4,0,0,4,0,0,290,0,10,33
9	0	
10	5	3,1,0,0,67
...	...	...

が共有される隣接ページはお互いに位置が近い．

- 類似性

お互いに位置が近い隣接ページは部分相続する隣接ページを共有する場合が多い．つまり，隣接リスト上の数字が似ている．

- 連続性

一つのページの隣接リスト上の数字は連続傾向がある．連続性はウェブページの構築特徴と上に述べた局所性と類似性によって引き起こされる．

3.2　*gap* の利用

表 1 と表 3 に示すように，Boldi と Vigna [1] はウェブグラフの連続性を利用し，*Successors* の差の絶対値をそれぞれ計算し，*gap* という手法を提案した．

ウェブグラフ $G = (V, E)$ のノード $v(v \in V)$ の隣接リストを以下の式

$$A_v = \{u_1, \ldots, u_k | u_i \in V\}$$

として表す．*gap* とは以下のようになる．

$$A_{gap}(v) = (u_1 - v, u_2 - u_1 - 1, u_3 - u_2 - 1, \cdots, u_k - u_{k-1} - 1) \quad (1)$$

辞書順で整列したため，$A_{gap}(v)$(式 1) の中に一番目の数式 "$u_1 - v$" 以外の数式の結果はすべて正数である．負数を防ぐために一番目の数式を以下のように処理する．一番目の数式の結果を x で表示する．

$$x = \begin{cases} 2(u_1 - v) & \{(u_1 - v) \geq 0\}, \\ 2 \mid u_1 - v \mid -1 & \{(u_1 - v) < 0\}, \end{cases}$$

すなわち，*gap* で表すウェブページ隣接構造は表 3 のように，隣接リスト上の数字を計算するので，最終的なリストに現れるすべての数字が小さくなる．また，ウェブグラフの連続性のため，10 以下の数字，特に "0" の出現回数が多い．

3.3　算 術 符 号

算術符号とは，各符号の出現確率を用い，符号化された系列全体を一つの符号語に符号化するものである．算術符号の利点は圧縮内容が長い文字列でも，数字列でも，圧縮結果は一つの数字になり，圧縮率が高いことである．算術符号は，長さの情報源系列とその出現確率が一対一に対応している．したがって，出現確率が分かれば，対応するデータを復元することが可能である．算術符号化の原理は以下に述べる [6]．圧縮するデータの出現率により，ある区間を逐次分割する．一般的に区間は [0,1) とする．したがって，データの出現率は分割した部分区間に対応する．対応させる部分区間の中に存在する値がデータを圧縮した結果である．

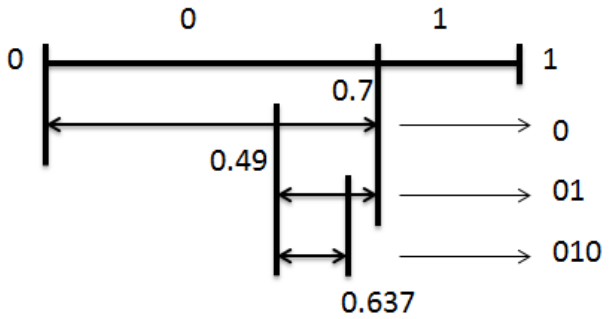

図 1　算術符号の例.

例えば，図 1 のように，入力データは {0,1} であり，"0" の確率を P(0)=0.7，"1" の確率を P(1)=0.3 として，データ列は "010" とする．まず，区間 [0,1) を "0" の確率 P(0)=0.7 と "1" の確率 P(1)=0.3 の比 7:3 を分割して，"0" は部分区間 [0,0.7)，"1" は部分区間 [0.7,1) に対応する．一番のデータは "0" であるため，部分区間 [0,0.7) が選択される．さらに，部分区間 [0,0.7) を 7:3 に分割して，"0" は部分区間 [0,0.49)，"1" は部分区間 [0.49,0.7) に対応する．二番のデータは "1" であるため，部分区間 [0.49,0.7) が選択される．同様に部分区間 [0.49,0.7) を 7:3 に分割して，"0" は部分区間 [0.49,0.637)，"1" は部分区間 [0.637,0.7) に対応する．最後のデータが "0" であるため，部分区間 [0.49,0.637) が選択される．すなわち，部分区間 [0.49,0.637) の中にある 0.49 が圧縮した結果と言える．

復号化の過程も複雑ではない．前の例において，"0" の区間は [0,0.7) であり，"1" の区間は [0.7,1) である．部分区間 [0.49,0.637) は "0" の区間の中にあるため，一番のデータは "0" である．その区間の下限値から "0" の区間の下限値を減じる結果と "0" の確率を割り算する．(0.49-0)/0.7=0.7 である．0.7 は "1" の区間 [0.7,1) の中にあるため，二番のデータは "1" である．同様に (0.7-0.7)/0.3=0 で，"0" の区間の中にあるため，最後のデータは "0" である．そこで，復号化の結果は "010" である．

4.　提 案 手 法

本稿では，各ページを自然数で符号化し，ウェブページの隣接リストを作成し，Boldi と Vigna の *gap* 手法と算術符号を組み合わせた圧縮手法を提案する．

4.1　圧 縮 過 程

数字を算術符号で復元するとき，出現した数字と対応出現率が必要である．隣接リスト上の数字はすべて違う．よって，すべての数字と出現率を保存しなければならず，データが膨大になる．隣接リスト上の数字は近い場合が多いため，隣接リスト上の自然数間の差の絶対値をそれぞれ計算すれば，結局はリストに現れるすべての数字が小さくなる．これらの数字を文字に変換し，順番に連結したものを算術符号で圧縮する．

手順 1.　各ノードに対応する隣接リストを整列

手順 2.　式 1 の数式のように，各隣接リスト上の自然数間の差をそれぞれ計算

手順 3.　手順 2 の結果の中から，各隣接リスト上の一番大きい

数字を抽出
手順 4.　手順 3 の結果それぞれの数字を文字列に変換
手順 5.　それぞれ隣接リスト上のすべての自然数を手順 4 の結果の文字列と同じ桁で文字列に変換
手順 6.　手順 5 の各ノードに対応する文字列を順番に連結
手順 7.　各ノードに対応する手順 6 の結果を算術符号でそれぞれ圧縮
［例 2］ 表 1 と表 3 に示すように，$Node7$ と $Node8$ を例として，圧縮する．
手順 1.　$Node7$ と $Node8$ の隣接リストを整列

$$A_7 = (8, 10, 11, 12, 13, 14, 18, 19, 198, 310, 2024)$$

$$A_8 = (10, 11, 12, 17, 18, 19, 310, 311, 322, 355)$$

手順 2.　式 1 の数式のように，各隣接リスト上の自然数間の差をそれぞれ計算

$$A_{gap(7)} = (2, 1, 0, 0, 0, 0, 3, 0, 178, 111, 1713)$$

$$A_{gap(8)} = (4, 0, 0, 4, 0, 0, 290, 0, 10, 33)$$

手順 3.　手順 2 の結果の中から，各隣接リスト上の一番大きい数字を抽出

$$A_{gap(7)max} = 1713$$

$$A_{gap(8)max} = 290$$

手順 4.　手順 3 の結果それぞれの数字を文字列に変換

$$String(A_{gap(7)max}) = 1713$$

$$String(A_{gap(8)max}) = 290$$

手順 5.　それぞれ隣接リスト上のすべての自然数を手順 4 の結果の文字列と同じ桁で文字列に変換

$String(A_{gap(7)}) =$

$$(0002, 0001, 0000, 0000, 0000, 0000, 0003, 0000, 0178, 0111, 1713)$$

$String(A_{gap(8)}) =$

$$(004, 000, 000, 004, 000, 000, 290, 000, 010, 033)$$

手順 6.　手順 5 の各ノードに対応する文字列を順番に連結

$String(A_{gap(7)})_{all} =$

$$00020001000000000000000000030000017801111713$$

$String(A_{gap(8)})_{all} =$

$$00400000000400000029000001003 3$$

手順 7.　各ノードに対応する手順 6 の結果を算術符号でそれぞれ圧縮

$$Result(A_{gap(7)}) = 0.24841949636786798$$

$$Result(A_{gap(8)}) = 0.4542699997115676$$

4.2 復号過程

隣接リストに戻す場合は，算術符号の復号過程のとおり，まず文字リストに戻り，それぞれ数字に変換し，各ノードの隣接リストに保存する．具体的な過程は以下である．
手順 1.　各ノードに対応する算術符号で圧縮結果を復号
手順 2.　手順 1 の結果を各ノードの $Outdegree$ の数字によって分ける
手順 3.　手順 2 の分けられた文字結果を数字に変換
手順 4.　手順 3 のそれぞれ結果を式 1 のように逆演算
手順 5.　手順 4 の結果をそれぞれ隣接リストに保存
［例 3］ 例 2 の結果を例として，表 1 と表 3 に示す $Node7$ と $Node8$ の隣接リストを戻す．
手順 1.　各ノードに対応する算術符号で圧縮結果を復号

$String'(A_{gap(7)})_{all} =$

$$00020001000000000000000000030000017801111713$$

$String'(A_{gap(8)})_{all} =$

$$00400000000400000029000001003 3$$

手順 2.　手順 1 の結果を各ノードの $Outdegree$ の数字によって分ける

$$Node7_{Outdegree} = 11$$

$$Node8_{Outdegree} = 10$$

よって，

$String'(A_{gap(7)}) =$

$$(0002, 0001, 0000, 0000, 0000, 0000, 0003, 0000, 0178, 0111, 1713)$$

$String'(A_{gap(8)}) =$

$$(004, 000, 000, 004, 000, 000, 290, 000, 010, 033)$$

手順 3.　手順 2 の分けられた文字結果を数字に変換

$$A'_{gap(7)} = (2, 1, 0, 0, 0, 0, 3, 0, 178, 111, 1713)$$

$$A'_{gap(8)} = (4, 0, 0, 4, 0, 0, 290, 0, 10, 33)$$

手順 4.　手順 3 のそれぞれ結果を式 1 のように逆演算

$$A'_7 = (8, 10, 11, 12, 13, 14, 18, 19, 198, 310, 2024)$$

$$A'_8 = (10, 11, 12, 17, 18, 19, 310, 311, 322, 355)$$

手順 5.　手順 4 の結果をそれぞれ隣接リストに保存

5. 実験と評価

提案した手法の性能を評価し，WebGraph，Re-pair，SSL と比較する．提案手法の実験環境は Intel(R) Core(TM) i5 2.53GHz 3M Cache 4GB RAM 上で行う．OS は Windows7 を用いる．実験データとしてベンチマークとして用いられる四つのデータ eu-2005，indochina-2004，uk-2002，arabic-2005

表 4　実験データ.

データ	ノード数	エッジ数	エッジ/ノード
eu-2005	862,664	19,235,140	22.30
Indochina-2004	7,414,866	194,109,311	26.18
uk-2002	18,520,486	298,113,762	16.10
Arabic-2005	22,744,080	639,999,458	28.14

表 5　実験データ eu-2005 の一部.

Node	*Outdegree*	*Successors*
1	752725	89607, 89608,89609,89610,89611...
2	4240	1,8063,8064,8065,8066...
...	...	...
862,664	1	1

表 6　WebGraph と比較結果.

データ	WebGraph	提案手法
eu-2005	7.3M	6.4M
Indochina-2004	26M	21M
uk-2002	65M	54M
Arabic-2005	106M	83M

[7](WebGraph のメインページ http://webgraph.dsi.unimi.it/ から取得した）を実験する．取得した四つの実験データはもう自然数で符号された．表 5 に実験データ eu-2005 の一部を示す．データ eu-2005 は小さいウェブグラフデータセットであり，データ indochina-2004，uk-2002，arabic-2005 はいろいろな分野から取得する大きいデータセットである．表 4 に四つのウェブデータを示す.

5.1　圧縮率の変化

表 6 のように，提案した手法と WebGraph の圧縮率 [8] を結果として比較し，圧縮率が良いという結果を得た．圧縮率について WebGraph のみと比較する．WebGraph で四つのデータサイズを圧縮した結果は [8] から取得した.

表 6 に示すように，提案手法と WebGraph [1] を比較し，圧縮率について提案手法の圧縮率は WebGraph より高い結果を得た.

5.2　アクセス時間の変化

圧縮したデータのサイズとアクセス時間の割合について WebGraph [1]，SSL [3] と Re-Pair [2]，三つの手法と比較する．図 2，3，4 と 5 には，三つの手法と圧縮したデータのサイズとアクセス時間の割合について比較した結果である．WebGraph と SSL の結果を [3] より抜粋し，Re-pair の結果を [2] より抜粋する．提案手法と WebGraph，SSL，Re-pair の実験環境は表 7 に示す．すべての手法のアルゴリズムは Java で実装されている．大部分の研究と同じように，我々は多くのランダムに選択された隣接リストを抽出し，エッジごとのアクセス時間を測定する．これらの時間を合計し，エッジ数によって合計時間で割る．スペースはエッジの総数によってデータサイズを分割し，データあたりのビット数で測定される.

図 2，3，4 と 5 の示すように，SSL は総体的に WebGraph より良いので，提案手法と SSL，Re-pair のみ比較する．圧縮率について提案手法と SSL，Re-pair と比べて高いことがわかる．また，アクセス時間について，図 2 の示すようにデータ eu-2005 のアクセス時間について提案手法は SSL より遅いが，図 3，4 と 5 の示すようにデータ Indochina-2004，データ uk-2002 とデータ Arabic-2005 について提案手法は SSL より速い．提案手法と Re-pair を比較するとアクセス時間が遅いことが推定できる.

表 7　実験環境.

手法	実験環境
提案手法	Intel Core i5 2.53GHz　3M Cache 4G RAM
WebGraph と SSL	Intel Core2 Q9450 2.66GHz 12M Cache 8G RAM
Re-pair	Intel(R) Xeon 2GHz　16GB RAM

5.3　実験評価

圧縮率について提案手法は WebGraph， SSL と Re-pair より高い．アクセス時間について実験環境が違うからアクセス時間の対比は厳密ではない．しかし，表 7 により，提案手法の実験環境が WebGraph， SSL の実験環境は大体同等程度である．また，データ Indochina-2004，データ uk-2002 とデータ Arabic-2005 のアクセス時間は提案手法で SSL より速いので，提案手法より大規模なデータのアクセス時間が WebGraph，SSL より速いことが推定できる．Re-pair と比べて，アクセス時間が推定できないが，提案手法の圧縮率は Re-pair よりずいぶん高いので，総体的に提案手法が良いことが言える.

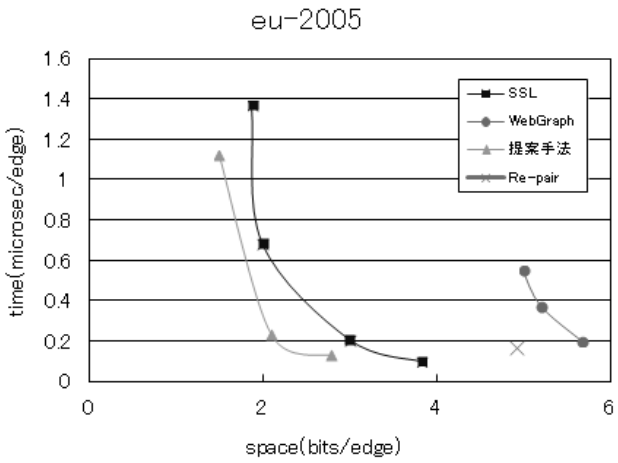

図 2　データ eu-2005 のアクセス時間対比.

6. ま と め

本稿では，*gap* と組み合わせ，算術符号を用いたウェブグラフの圧縮方法を提案した四つのデータ eu-2005，indochina-2004，uk-2002，arabic-2005 を利用し，実験の結果，提案手法で圧縮率が別に 53.3 %，43.75 %，50.47 %，38.79 %であり，WebGraph，Re-pair と SSL より高い結果を得た．また，圧縮したデータのサイズとアクセス時間の割合について WebGraph，

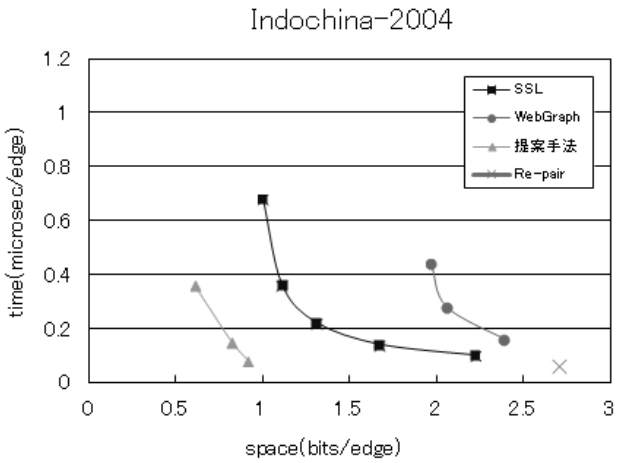

図 3　データ Indochina-2004 のアクセス時間対比.

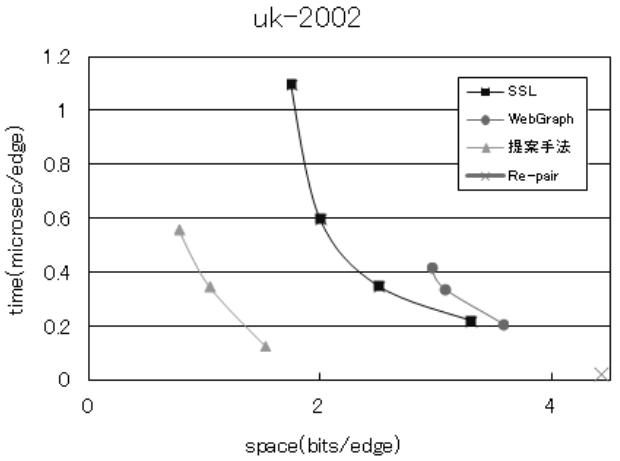

図 4　データ uk-2002 のアクセス時間対比.

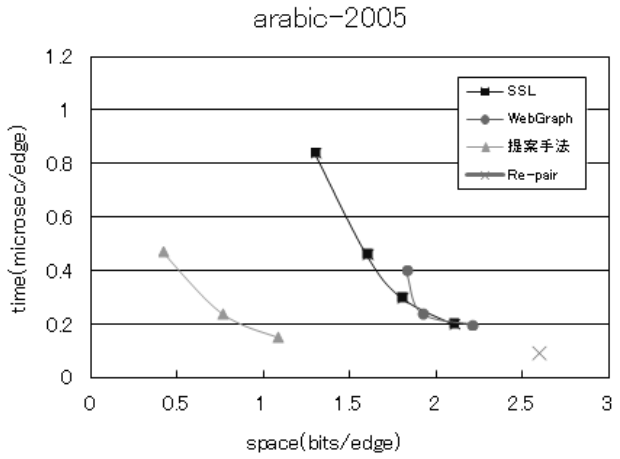

図 5　データ Arabic-2005 のアクセス時間対比.

Re-pair と SSL の結果は論文からの引用であり，実験環境は多少異なるものの，提案の手法は WebGraph，Re-pair，SSL と比較し，大規模なデータの処理性能が良い考えられる．

しかし，算術符号で圧縮する処理時間が少し長いという欠点があり，実際に応用の場合，今後改良する必要がある．また，様々なウェブグラフを用いた検証も必要である。将来の展望として，一つの隣接リストだけではなく，より多い隣接リストの間の関係を利用することにより，性能を向上させ，圧縮率をより改善できることが期待される．

文　献

[1] Boldi. P, Vigna. S, “The webgraph framework I: compression techniques”, in ’Proceedings of the 13th conference on World Wide Web’ , ACM Press, New York, NY, USA , pp. 595–602 (2004).

[2] Claude. F, Navarro. G, “Fast and Compact Web Graph Representations”, TWEB 4 (4) (2010).

[3] S. Grabowski, W. Bieniecki, “Tight and simple Web graph compression for forward and reverse neighbor queries”, Discrete Applied Mathematics ,Volume 163 ,Part 3 ,pp.298-306 (2013).

[4] Buehrer. G, Chellapilla. K, “A scalable pattern mining approach to web graph compression with communities”, in ’WSDM ’08: Proceedings of the international conference on Web search and web data mining’ , ACM, New York, NY, USA , pp. 95–106 (2008).

[5] Navlakha. S, Rastogi. R, Shrivastava. N, “Graph summarization with bounded error”, in Jason Tsong-Li Wang, ed., ’SIGMOD Conference’ , ACM, , pp. 419-432 (2008).

[6] G. G. Langdon, “ An Introduction to Arithmetic Coding”, IBM Journal of Research and Development 28 (2) , pp.135-149 (1984).

[7] Boldi. P, Codenotti. B, Santini. M, Vigna. S, UbiCrawler,“ a scalable fully distributed Web crawler”, Softw., Pract. Exper. 34 (8) , pp. 711-726 (2004).

[8] http://law.di.unimi.it/datasets.php

[9] Boldi. P, Vigna. S, “The WebGraph Framework II: Codes For The World-Wide Web”, in ’Data Compression Conference’ , IEEE Computer Society, , pp. 528 (2004).

[10] L. Page, S. Brin. R. Motowani, T. Winograd, “The PageRank Ciation Ranking: Bringing Order to the Web”, Technical report, Stanford Digital Libraries, (1998).

[11] Lempel. R, Moran. S, “The stochastic approach for link-structure analysis (SALSA) and the TKC effect”, Computer Networks 33 (1-6) , 387-401 (2000).

[12] S. R. Kurmar, P. Raphavan, S. Rajagopalan, A. Tomkins,“Trawling the Web for emerging cyger communities” , In Proc. of 8^{th}Int. WWW Conf., (1999).

[13] Broder. A. Z, Charikar. M, Frieze. A. M, Mitzenmacher. M, “Min-Wise Independent Permutations”, J. Comput. Syst. Sci. 60 (3) , pp. 630-659 (2000).