IMPLEMENTASI TEKNIK KOMPRESI TEKS HUFFMAN

Andysah Putera Utama Siahaan
Fakultas Ilmu Komputer
Universitas Pembanguan Panca Budi
Jl. Jend. Gatot Subroto Km. 4,5 Sei Sikambing, 20122, Medan, Sumatera Utara, Indonesia
andiesiahaan@gmail.com

Abstrak

Huffman adalah salah satu algoritma kompresi. Ini adalah algoritma paling terkenal untuk kompres teks. Ada empat fase dalam algoritma Huffman untuk kompres teks. Yang pertama adalah kelompok karakter. Yang kedua adalah untuk membangun pohon Huffman. Yang ketiga adalah encoding, dan yang terakhir adalah pembangunan bit kode. Prinsip algoritma Huffman adalah karakter yang sering muncul di encoding dengan rangkaian bit yang pendek dan karakter yang jarang muncul di bit-encoding dengan seri lagi. teknik kompresi Huffman dapat memberikan penghematan dari 30% dari bit asli. Ia bekerja berdasarkan frekuensi karakter. Semakin banyak karakter yang sama mencapai, semakin tinggi tingkat kompresi yang diperoleh.

Kata Kunci — Huffman, Compression, Algorithm, Security

I. Pendahuluan

Sebuah teks adalah kumpulan karakter atau string menjadi satu kesatuan. Ini berisi banyak karakter di dalamnya yang selalu menimbulkan masalah di perangkat penyimpanan yang terbatas dan kecepatan transmisi data pada waktu tertentu. Meskipun penyimpanan dapat digantikan oleh yang lain lebih besar, ini bukan solusi yang baik jika kita masih bisa menemukan solusi lain. Dan ini membuat semua orang mencoba untuk berpikir untuk menemukan cara yang dapat digunakan untuk kompres teks.

Kompresi adalah proses mengubah data asli ke bentuk kode untuk menghemat penyimpanan dan waktu persyaratan untuk transmisi data. Sebuah algoritma kompresi loseless harus menekankan orisinalitas data selama proses kompresi dan dekompresi [1]. Dengan menggunakan algoritma Huffman, proses kompresi teks dilakukan dengan menggunakan prinsip pengkodean; masing-masing karakter dikodekan dengan rangkaian beberapa bit untuk menghasilkan hasil yang lebih optimal. Tujuan penulisan makalah ini adalah untuk mengetahui efektivitas dan cara terpendek algoritma Huffman dalam kompresi teks dan menjelaskan cara mengkompresi teks menggunakan algoritma Huffman dalam pemrograman.

II. Landasan Teori

Kompresi data adalah proses menyusut data ke bit lebih kecil dari representasi asli sehingga dibutuhkan lebih sedikit ruang penyimpanan dan waktu transmisi kurang saat berkomunikasi melalui jaringan [2]. Huffman algoritma diciptakan oleh seorang mahasiswa MIT bernama David Huffman pada tahun 1952. Ini adalah salah satu metode tertua dan paling terkenal di kompresi teks [4]. Kode Huffman menggunakan prinsip-prinsip yang mirip dengan kode Morse. Setiap karakter dikodekan hanya dengan rangkaian beberapa bit, di mana karakter yang sering muncul dengan serangkaian kode bit pendek dan karakter yang jarang muncul dikodekan dengan panjang set bit. Berdasarkan jenis peta, kode ini digunakan untuk mengubah pesan awal (isi input data) menjadi satu set codeword. Algoritma Huffman menggunakan metode statis. Sebuah metode statis adalah metode yang selalu menggunakan peta kode yang sama tapi kita masih bisa mengubah urutan penampilan karakter. Metode ini membutuhkan dua langkah. Langkah pertama untuk menghitung frekuensi terjadinya masing-masing simbol dan menentukan kode peta, dan yang terakhir adalah untuk mengkonversi pesan menjadi kumpulan kode yang akan ditransmisikan. Sementara itu,

berdasarkan pada simbol teknik coding, Huffman menggunakan metode symbolwise. Symbolwise adalah metode yang menghitung frequncy karakter dalam setiap proses. Tranforming karakter teks ke symbolwise bukanlah proses yang mudah [3]. Simbol ini lebih sering terjadi akan diberikan kode lebih pendek dari simbol-simbol yang jarang muncul.

A. Algoritma Greedy

Algoritma Greedy memecahkan masalah dengan memilih jarak terbaik pada saat tertentu. masalah Huffman dapat diselesaikan dengan menggunakan algoritma serakah serta [5]. Sebuah algoritma serakah adalah suatu algoritma yang mengikuti metaheuristik pemecahan masalah dari membuat pilihan yang optimal. Dengan menghitung setiap langkah, kami berharap kami dapat menemukan solusi yang optimal. Misalnya, menerapkan strategi serakah dengan TSP untuk mengunjungi tempat yang belum dikunjungi terdekat. algoritma serakah tidak pernah menemukan solusi global. algoritma ini baik dalam menemukan solusi terdekat. Dua contoh dari algoritma serakah yang Algoritma Kruskal dan Prim.

B. Algoritma Huffman.

David Huffman dikodekan karakter dengan hanya menggunakan pohon biner biasa, tapi setelah itu, David Huffman menemukan bahwa menggunakan algoritma serakah dapat membangun kode prefix optimal. Penggunaan algoritma serakah pada Huffman adalah pada pemilihan dua pohon dengan frekuensi terendah di pohon Huffman. Sebuah algoritma serakah digunakan untuk meminimalkan total biaya. Biaya ini digunakan untuk menggabungkan dua pohon pada akar frekuensi sama dengan jumlah dua pohon buah yang digabungkan. Oleh karena itu total biaya pembentukan pohon Huffman adalah total dari seluruh merger. Oleh karena itu, algoritma Huffman adalah salah satu contoh dari algoritma kompresi yang menggunakan algoritma serakah. Tujuan kami adalah untuk menghitung total biaya yang dikeluarkan untuk membangun teks.

III. Evaluasi

Dalam implementasinya, kami mencoba untuk mencari tahu teknik Huffman. Ada empat langkah yang harus dilakukan untuk membuat teknik Huffman dioperasikan sepenuhnya. Fase-fase ini di bawah menjelaskan langkah-langkah dari algoritma.

A. FasePertama.

Asumsikan bahwa kita memiliki kalimat "Lika-LIKU LAKI-LAKI TAK LAKU-LAKU". Teks di atas adalah 33 panjang. Itu harus dikategorikan dasar pada frekuensi karakter. Fungsi character_set adalah untuk menentukan berapa kali masing-masing karakter muncul.

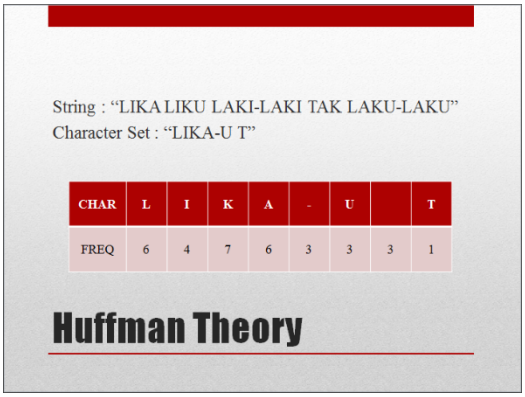
```
function Character_Set(text : string) : string;
var
  temp   : string;
  result : string;

begin
  temp := text;
  result := '';

  for i := 1 to length(temp) do
    for j := i + 1 to length(temp) do
      if temp[i] = temp[j] then temp[j] := '#';

  j := 1;

  for i := 1 to length(temp) do
    if temp[i] <> '#' then
    begin
      result := concat(result, temp[i]);
      inc(j);
    end;
```

```
  Character_Set := result;
end;
```

Loop pertama yang dilakukan oleh "untuk" adalah untuk menggantikan duplikat karakter dalam '#'. Loop kedua menghapus teks asli yang terdiri dari '#'. Kita bisa lihat ilustrasi di bawah ini.

```
Original Text : LIKA-LIKU LAKI-LAKI TAK LAKU-LAKU
Replaced Text : LIKA-###U ##########T#############
Character Set : LIKA-U T

procedure Character_Freq(text : string);
var
  temp  : string;
  freq  : byte;

begin
  temp := Character_Set(text);

  for i := 1 to length(temp) do
  begin
    freq := 0;
    for j := 1 to length(text) do
      if temp[i] = text[j] then inc(freq);
    AddNode(Head, Tail, temp[i], freq);
  end;
end;
```

Prosedur di atas menghitung frekuensi terjadinya karakter. Pertama, kita harus menjalankan fungsi set karakter untuk mendapatkan serangkaian karakter tunggal yang digunakan; maka kita harus membandingkan dari karakter pertama sampai karakter terakhir pada teks asli untuk mendapatkan frekuensi. Karena kita menggunakan pointer untuk mewakili nilai. Hasilnya akan dikirim ke node setelah mendapatkan bertambah. Gambar 1 menggambarkan karakter dan frekuensi mereka memperoleh dari Fase Satu.



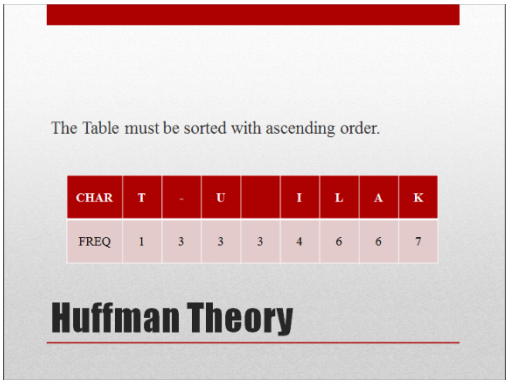
Gambar 1. Frekuensi kemunculan karakter

Tabel harus diurutkan dalam urutan menaik dan kunci utama adalah frekuensi.

```
procedure Tree_Sorting;
var
  tASCII  : char;
  tFreq   : byte;

begin
  for i := length(cs) - 1 downto 1 do
  begin
    Current := Head;

for j := 1 to i do
    begin
      if Current^.Freq > Current^.Next^.Freq then
```

```
        begin
          tASCII := Current^.ASCII;
          tFreq  := Current^.Freq;
          Current^.ASCII := Current^.Next^.ASCII;
          Current^.Freq  := Current^.Next^.Freq;
          Current^.Next^.ASCII := tASCII;
          Current^.Next^.Freq  := tFreq;
        end;
        Current := Current^.Next;
      end;
    end;
  end;
```

Prosedur di atas adalah untuk mengurutkan daftar disortir dengan menggunakan algoritma bubble sort. Node pertama akan dibandingkan dengan node berikutnya, dan nilai yang lebih besar akan bertukar dan pindah ke kanan. Kami memiliki nilai terkecil di sebelah kiri. Gambar 2 menunjukkan hasil setelah semacam itu.



Gambar 2. Frekuensi karakter setelah terurutkan

B. FaseKedua

Setelah tabel sepenuhnya diurutkan, sekarang saatnya untuk menghadapi langkah yang paling sulit; yang membuat pohon Huffman. Kita harus mengkategorikan setiap node dan meletakkannya di linked list. Algoritma Greedy mengambil terpisah di bagian ini. Kita harus menggabungkan dua node dan membuat node baru dan menjadikannya sebagai induk dari simpul sebelumnya. Mari kita lihat ilustrasi di bawah ini:

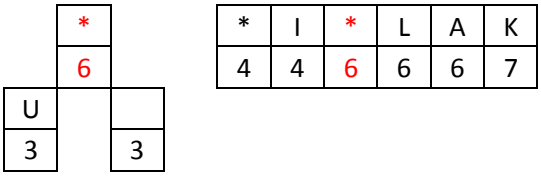
T	-	U		I	L	A	K
1	3	3	3	4	6	6	7

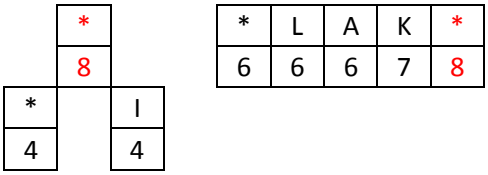
Dua node pertama akan diambil dan dilepaskan dari tabel. Membuat node baru yang akan menjadi orang tua mereka.

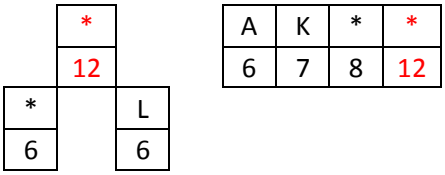
	*	
	4	
T		-
1		3

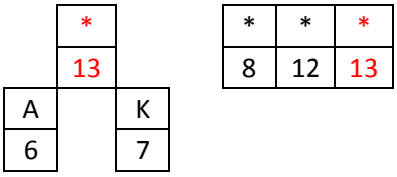
U		*	I	L	A	K
3	3	4	4	6	6	7

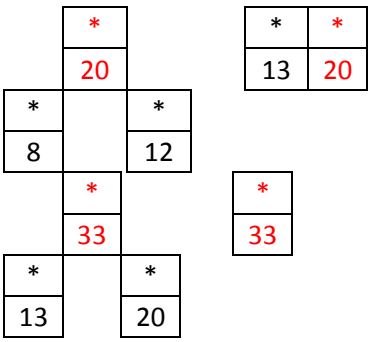
Induk simpul akan dimasukkan ke tabel dalam urutan menaik menggunakan algoritma penyisipan. Simpul pertama kini digantikan oleh node ketiga sebelum. Selain itu, kita harus melakukan hal yang sama lagi sampai tabel terdiri dari satu simpul.







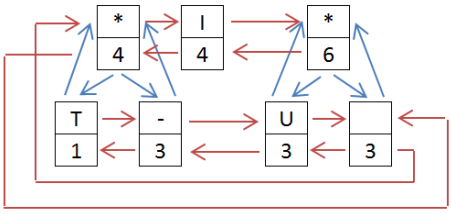




T	-	U		*	I	L	A	K						
1	3	3	3	4	4	6	6	7						
T	-	U		*	I	*	L	A	K					
1	3	3	3	4	4	6	6	6	7					
T	-	U		*	I	*	L	A	K	*				
1	3	3	3	4	4	6	6	6	7	8				
T	-	U		*	I	*	L	A	K	*	*			
1	3	3	3	4	4	6	6	6	7	8	12			
T	-	U		*	I	*	L	A	K	*	*	*		
1	3	3	3	4	4	6	6	6	7	8	12	13		
T	-	U		*	I	*	L	A	K	*	*	*	*	
1	3	3	3	4	4	6	6	6	7	8	12	13	20	
T	-	U		*	I	*	L	A	K	*	*	*	*	*
1	3	3	3	4	4	6	6	6	7	8	12	13	20	33

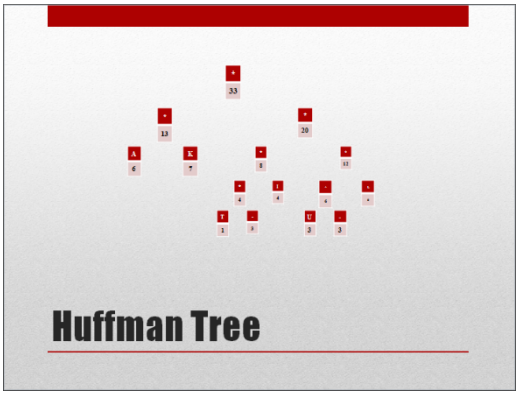
Array terdiri dari 15 nodes.

Tahap ini memiliki dua jenis linked list:
- Double Linked List.
- Binary Tree Linked List.



Gambar 3.Double & Binary Tree Linked List

Gambar 3 menunjukkan hirarki dari array. * menunjukkan node memiliki orangtua. Gambar 4 adalah diagram lengkap dari pohon Huffman.



Gambar4.Pohon Huffman

Tujuan menggunakan dua model linked list adalah untuk menghindari prosedur pencarian. Kita dapat menggunakan breadth-first atau kedalaman-pertama pencarian untuk membentuk kode bit, tetapi membutuhkan waktu dan teknik pemrograman tingkat lanjut. Namun, kita dapat menggunakan daftar untuk menggantikan prosedur backtracking. Kami hanya memiliki untuk melacak siapa orang tua adalah.

```
procedure Huffman_Tree;
var
  tFreq   : byte;
  tASCII  : char;
  InsertN : NodeP;
  NewNode : NodeP;

begin
  tASCII := '*';
  Current := Head;

  while Current^.Next <> NIL do
  begin
    InsertN := Head;
    tFreq := Current^.Freq + Current^.Next^.Freq;
    Current^.Bit       := 0;
    Current^.Next^.Bit := 1;

    while InsertN <> NIL do
    begin
      if tFreq <= InsertN^.Freq then
```

```
      begin
        new(NewNode);
        NewNode^.Freq := tFreq;
        NewNode^.ASCII := tASCII;
        NewNode^.Next := InsertN;
        NewNode^.Prev := InsertN^.Prev;
        InsertN^.Prev^.Next := NewNode;
        InsertN^.Prev := NewNode;

        NewNode^.Left   := Current;
        NewNode^.Right  := Current^.Next;
        Current^.Parent := NewNode;
        Current^.Next^.Parent := NewNode;
        break;
      end
      else if tFreq > Tail^.Freq then
      begin
        new(NewNode);
        NewNode^.Freq := tFreq;
        NewNode^.ASCII := tASCII;
        Tail^.Next := NewNode;
        NewNode^.Prev := Tail;
        Tail:=NewNode;
        Tail^.Next := NIL;

        NewNode^.Left   := Current;
        NewNode^.Right  := Current^.Next;
        Current^.Parent := NewNode;
        Current^.Next^.Parent := NewNode;
        break;
      end;
      InsertN := InsertN^.Next;

    end;

    Current := Current^.Next;
    Current := Current^.Next;
  end;
end;
```

Prosedur Huffman digunakan untuk membentuk pohon Huffman dengan mengolah pohon linear sebelumnya. Kita harus menandai node "yang berada di sebelah kiri" dan "yang ada di sebelah kanan" dengan menambahkan tanda 0 atau 1 untuk bidang simpul.

```
type
  NodeP     = ^Node;
  Node      = record
                ASCII  : char;
                Bit    : byte;
                Code   : string;
                Dec    : byte;
                freq   : byte;
                Prev,
                Next   : NodeP;
                Parent,
                Left,
                Right  : NodeP;
              end;
```

ASCII	:	where character is memorized.
Bit	:	node sign. (left or right)
Code	:	Huffman code.
Dec	:	decimal code.
Freq	:	occurence of the character.
Next, Prev, Parent, Left, Right	:	represent the connected nodes.

Dua node pertama harus dikombinasikan. Node pertama akan ditandai sebagai '0' karena itu adalah di sebelah kiri dan simpul kedua akan ditandai sebagai '1' karena itu adalah salah satu kanan. Selain itu, kedua node memiliki induk yang sama, dan orang tua memiliki dua anak, node. Setelah node induk dibuat, harus dimasukkan ke dalam daftar link dengan menggabungkan nilai node dan nilai linked list. node harus ditambahkan di sebelah kiri nilai yang lebih besar atau sama. Namun, jika node induk lebih besar dari setiap node, itu harus dimasukkan setelah node terakhir.

C. Fase Ketiga

Pada langkah ini, pohon tersebut sudah terstruktur. Ini adalah waktu untuk mengambil tanda simpul dengan membuat lingkaran sampai induk dari node terakhir kosong (null).

```
procedure Write_Huffman;
var
  result : string;
  bit    : string[1];

begin
  Current := Head;

  repeat
    result := '';
    Cursor := Current;
    Current^.Dec := 0;
    Biner := 1;

    if Cursor^.ASCII <> '*' then
    begin
      repeat
        if (Cursor^.Bit = 0) or
(Cursor^.Bit = 1) then


        begin
          Current^.Dec := Current^.Dec +
 (Cursor^.Bit * Biner);
          Biner := Biner * 2;
          str(Cursor^.Bit, Bit);
          insert(Bit, result, 1);
        end;
        Cursor := Cursor^.Parent;
      until Cursor^.Parent = NIL;
    end;

    Current^.Code := result;
    Current := Current^.Next;
  until Current^.Next = NIL;
end;
```

Selain itu, tanda simpul dari node akan dimasukkan ke dalam sebuah string tunggal.

D. Fase Keempat

Setiap node telah terdiri dari kode. Fase ini, segala sesuatu yang telah dikodekan dikonversi ke tabel Huffman. Tabel 1 menunjukkan prioritas karakter. Karakter "K" memiliki paling penampilan sementara karakter "T" memiliki kurang satu. Karakter dengan munculnya tertinggi adalah memiliki kode biner terpendek.

Table 1. Huffman Table.

Char	Freq.	Code	Bit Len.	Code Len.

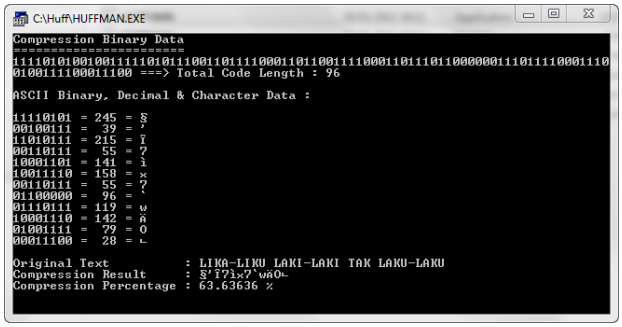
T	1	1000	4	4
-	3	1001	4	12
U	3	1100	4	12
	3	1101	4	12
I	4	101	3	12
L	6	111	3	18
A	6	00	2	12
K	7	01	2	14
				96

Setiap kode mewakili karakter. Ini terdiri dari beberapa digit dari string biner. Teks sebelumnya akan berubah menjadi set biner baru. The berusia delapan bit diganti dengan yang baru. Mari kita lihat contoh di bawah ini:

```
Original Text : LIKA-LIKU LAKI-LAKI TAK LAKU-LAKU
Original Code :

111 101 01 00 1001 111 101 01 1100 1101 111 00 01 101 1001 111 00 01 101 1101 1000 00 01
1101 111 00 01 1100 1001 111 00 01 1100

Bit Code :

11110101 00100111110101110011011110001101
10011110001101110110000 01110111100011 10
01001111 00011100

Decimal Code :

254, 39, 215, 55, 141, 158,
55, 96, 119, 142, 79, 28
```

Gambar 5. Running Program (Bagian. 1).

Gambar 6. Running Program (Bagian. 2).



Gambar 7. Running Program (Bagian. 3)

Gambar 5 ke Gambar 7 adalah proses Huffman berjalan. Teks aslinya sebelum kompresi mengambil 33 karakter panjang. Cepat setelah dikompresi, string hanya membutuhkan 12 karakter. Jadi kita menghemat 21 karakter.

```
Original Text Length   :  33
Coded Text Length      :  12
Saving Rate            :  (33 - 12) / 33 * 100 %
                          63.63636 %
```

Dengan melakukan algoritma ini, tingkat kompresi mencapai 63% dari pesan asli. Tentu saja, itu akan menghemat kapasitas penyimpanan.

IV. Kesimpulan

Penelitian ini menjelaskan teknik dasar tentang cara menerapkan algoritma kompresi. algoritma Huffman dapat menggabungkan dengan algoritma serakah yang selalu menemukan cara termudah dari node terdekat. Ada empat fase dalam algoritma Huffman untuk kompres teks; yang pertama adalah fase untuk mengelola dan mendapatkan frekuensi karakter. Yang kedua adalah pembentukan pohon Huffman; yang ketiga adalah untuk membentuk kode dari tanda simpul. Selain itu, tahap terakhir adalah proses encoding. Namun, dalam makalah ini, kami hanya melakukan metode encoding, decoding berada di bawah proyek. Menerapkan Huffman dalam kompresi akan lagi lebih banyak ruang dalam penyimpanan. Huffman encoding sangat ampuh untuk pesan teks yang memiliki terjadinya karakter yang sama.

DAFTAR PUSTAKA

[1] I. Akman, H. Bayindir, S. Ozleme, Z. Akin and a. S. Misra, "Lossless Text Compression Technique Using Syllable Based Morphology," *International Arab Journal of Information Technology,* vol. 8, no. 1, pp. 66-74, 2011.

[2] A. S. Sidhu and M. Garg, "Research Paper on Text Data Compression Algorithm using Hybrid Approach," *IJCSMC,* vol. 3, no. 12, pp. 1-10, 2014.

[3] H. Al-Bahadili and S. M. Hussain, "A Bit-level Text Compression Scheme Based on the ACW Algorithm," *International Journal of Automation and Computing,* pp. 123-131, 2010.

[4] M. Schindler, "Practical Huffman coding," 1998. [Online]. Available: http://www.compressconsult.com/huffman/.

[5] A. Malik, N. Goyat and V. Saroha, "Greedy Algorithm: Huffman Algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 3, no. 7, pp. 296-303, 2013.