Analisis Algoritma MTF, MTF-1 dan MTF-2 pada Burrows Wheeler Compression Algorithm

Sagara Mahardika Sunaryo[1], Lukas Chrisantyo[2], Yuan Lukito[3]
Program Studi Informatika, Universitas Kristen Duta Wacana Yogyakarta
[1]sagara.mahardika@ti.ukdw.ac.id
[2]lukaschris@ti.ukdw.ac.id
[3]yuan@ti.ukdw.ac.id

Abstract*—The need for text data compression in the era of cloud computing is still quite high. Text data needs to be compressed as small as possible to be easily sent. Burrows Wheeler Compression Algorithm (BWCA) is a type of block sorting compression algorithm that is non-proprietary and is quite popular among other text compression algorithms. In the process, BWCA uses an initial processing method called Global Structure Transformation (GST) to arrange characters for better compression result. This study compared three Move-to-Front pre-processing methods, namely MTF, MTF-1 and MTF-2. Test materials are taken from English, Indonesian and Javanese Bible texts, and some testing data from the Calgary Corpus. Since text compression is a lossless and reversible compression, in addition to the whole process, researcher has also performed a decompression phase with Inverse Burrows Wheeler Transform. The result showed that MTF-1 method was able to provide a better compression ratio because the total number of each bit in the Huffman process was less than the other two methods.

Intisari—Kebutuhan kompresi data teks di era komputasi awan saat ini masih cukup tinggi. Data teks perlu dikompresi sekecil mungkin agar mudah dikirimkan. Burrows Wheeler Compression Algorithm (BWCA) adalah salah satu algoritma kompresi teks jenis block sorting yang bersifat non-proprietary dan cukup populer digunakan. Dalam prosesnya, BWCA menggunakan metode pemrosesan awal yang disebut Global Structure Transformation (GST) untuk menyusun karakter agar lebih baik hasil kompresinya. Penelitian ini membandingkan tiga metode pemrosesan awal Move-to-Front, yaitu MTF, MTF-1 dan MTF-2. Bahan uji kompresi berupa data Alkitab Bahasa Inggris, Indonesia dan Jawa, dan beberapa data yang berasal dari Calgary Corpus. Oleh karena kompresi teks adalah kompresi yang bersifat lossless dan reversibel, maka selain melakukan pengujian untuk pengompresian data, juga dilakukan pengujian untuk pendekompresian data dengan Inverse Burrows Wheeler Transform. Pengujian kompresi dan dekompresi pada data Alkitab maupun Calgary Corpus berhasil dilakukan dan menunjukkan MTF-1 mampu memberikan rasio kompresi yang lebih baik dikarenakan jumlah total tiap bit pada proses Huffman lebih sedikit dibandingkan dua metode lainnya.

***Kata Kunci*— Move-to-Front, Burrows Wheeler Compression, kompresi data**

I. PENDAHULUAN

Kompresi data adalah sebuah seni atau teknik dalam merepresentasikan informasi ke dalam bentuk yang lebih kompleks. Menurut Sayood [1] kompresi data mampu menciptakan bentuk yang lebih kompleks dengan mengidentifikasi dan menggunakan struktur yang ada di dalam data. Data di sini memiliki banyak jenis, yaitu teks, *file*, bentuk-bentuk angka dari sampel *audio*/suara dan *image*/citra. Kompresi data masih ada dan terus digunakan karena masih memegang peranan vital dalam pertukaran informasi melalui internet, penyimpanan system informasi digital, penggunaan system *embedded*, dan pengiriman *file*.

Algoritma kompresi teks cukup beragam. Salah satunya adalah Burrows Wheeler Compression Algorithm (BWCA) yang merupakan algoritma *block sorting* yang cukup popular dan tidak dilindungi hak cipta (*non-proprietary*), yang berarti algoritma ini boleh digunakan atau dimodifikasi oleh siapapun. Di dalam proses BWCA terdapat empat tahap, yaitu Burrows-Wheeler Transform (BWT), Global Structure Transformation (GST), Run Length Encoding (RLE) dan yang terakhir Entropy Coding (EC).

Baruah meneliti tentang perbandingan algoritma BWCA murni dengan yang sudah dimodifikasi. Dalam modifikasinya, Baruah menambahkan satu tahap RLE antara BWT dan GST. Hasil yang didapatkan BWCA yang murni masih menghasilkan nilai rata-rata rasio kompresi yang lebih baik daripada yang dimodifikasi [2].

Anderson dan Nilsson mencoba untuk memodifikasi algoritma BWCA dengan membuat perubahan pada proses pengurutan algoritma BWT dengan algoritma *Radix Sort*. Hsail penelitian ini didapatkan rasio kompresi yang sama namun waktu kompresi menjadi lebih cepat [3]. Kemudian Itoh dan Tanaka [4] membuat perubahan pada proses pengurutan algoritma BWT dengan algoritma *Itoh Tanaka Suffix Array*. Hasil dari penelitian ini tetap memberikan rasio kompresi yang sama, namun waktu kompresi lebih cepat daripada menggunakan *Radix Sort*.

Nanda dalam penelitiannya mengatakan bahwa modifikasi Move-to-Front yang dalam papernya disebut sebagai Advanced MTF (AMTF) menyimpulkan bahwa beban yang dibutuhkan AMTF lebih sedikit dibandingkan

menggunakan MTF yang asli untuk kebutuhan akses daftar elemen-elemen yang letaknya berjauhan [5]

Dalam penelitian ini akan dicoba untuk memodifikasi tahap kedua BWCA yaitu tahap GST dengan tujuan untuk menganalisis algoritma manakah yang dapat memberikan rasio kompresi yang lebih baik, apakah MTF yang asli, atau modifikasi pertama (MTF-1) atau modifikasi kedua (MTF-2) untuk seluruh data uji yang digunakan.

Data uji menggunakan teks Alkitab dalam Bahasa Indonesia, Inggris dan Jawa serta data uji standar kompresi dari Calgary Corpus, yaitu bib, book, news, paper1, paper2, paper3, paper4, paper5, paper6, progc, progl, progp.

II. LANDASAN TEORI

Metode yang diterapkan dalam penelitian ini antara lain Burrows Wheeler Transform, Move-to-Front dan modifikasinya, Run Length Encoding & Decoding 0, Huffman Encoding & Decoding, dan Inverse MTF serta Inverse BWT. Metode Huffman sudah cukup dikenal dalam dunia kompresi data.

A. Burrows Wheeler Transform

Menurut Deorowicz [6], Burrows Wheeler Transform (BWT) merupakan bentuk transformasi suatu kata menjadi kata lain melalui proses pembentukan *Suffix Array* (SA). Pembentukan SA mempunyai 4 variabel di dalam proses tersebut, yaitu sebagai berikut :

1. Variabel sa digunakan untuk menyimpan hasil suffix array yang diurutkan secara alfabet dengan dibatasi variabel gap.
2. Variabel gap digunakan untuk membatasi pengurutan secara alfabet variabel sa.
3. Variabel order digunakan untuk menentukan peringkat tiap suffix.
4. Variabel temp digunakan untuk menampung suffix array pada iterasi sebelumnya.

Pembentukan SA merupakan sebuah iterasi, dimana proses untuk memberhentikan iterasi tersebut bergantung pada setiap suffix yang telah dibuat pada iterasi tersebut yang tidak sama lagi atau unik. Berikut tiga proses utama tiap iterasi dalam proses pembentukan SA:

1. Proses pertama adalah proses pengurutan. Disini penulis memakai *Merge Sort* sebagai proses pengurutannya, sedangkan untuk proses pertukaran pada saat pengurutan, penulis akan menggunakan algoritma SA khusus pertukaran.
2. Proses kedua adalah proses ranking. Pada proses ini, setiap value pada SA dibandingkan dengan tetangganya. Apabila mereka berdua identik / sama maka tidak ada pertambahan ranking, apabila mereka berdua berbeda maka tambah ranking pada saat ini + 1. Cara menentukan mereka berdua identik atau tidak, menggunakan algoritma SA khusus pertukaran.
3. Proses ketiga adalah proses peletakkan tiap huruf/angka sesuai dari proses ranking, untuk membentuk SA selanjutnya.

Setelah proses pembentukan SA telah selesai. BWT harus diambil dengan cara menyimpan setiap data pada kedalam variabel baru dengan menggunakan SA yang telah dibentuk sebelumnya.

Contoh input: banana$

1. Isi pembentukan SA suffix array → 0,1,2,3,4,5,6, dan isi array temp → 98,97,110,97,110,97,36.
2. Sort SA dengan menggunakan array temp. Isi SA setelah sort → 6,5,3,1,0,4,2.
3. Beri peringkat SA dengan menggunakan algoritma pertukaran SA. Isi array order setelah proses pemeringkatan → 0,1,2,2,3,4,4.
4. Proses peletakkan tiap value SA sesuai dengan ranking pada proses sebelumnya yang dimasukkan pada array temp. Isi array temp setelah proses ini → 3,2,4,2,4,1,0
5. Karena value pada array order paling terakhir != 7 – 1 (panjang data -1), maka tiga proses pembentukan SA, diulangi lagi.
6. Sort SA dengan menggunakan array temp. Isi SA setelah sort → 6,5,3,1,0,4,2.
7. Beri peringkat SA dengan menggunakan algoritma pertukaran SA. Isi array order setelah proses pemeringkatan → 0,1,2,3,4,5,6.
8. Proses peletakkan tiap value suffix array sesuai dengan ranking pada proses sebelumnya yang dimasukkan pada array temp. Isi array temp setelah proses ini → 4,3,6,2,5,1,0.
9. Karena value pada array order paling terakhir == 7 – 1 (panjang data -1), maka proses pembentukan SA telah selesai.
10. Ambil resultBwt, dengan menggunakan suffix array. resultBwt → annb$aa.

B. Move-to-Front

Global Structure Transformation adalah tahap setelah BWT yang berfungsi untuk mengubah rangkaian huruf yang dihasilkan oleh BWT menjadi rangkaian index [2]. Algoritma yang biasa dipakai oleh BWCA adalah MTF. Cara kerja dari algoritma MTF dengan inputan yang berasal dari variabel **resultBwt**:

1. Simpan rangkaian simbol/huruf dari ASCII 0 - 127 pada variabel **mtf**.
2. Lakukan pencarian index dalam variabel **mtf**, dari huruf-huruf yang berasal dari **resultBwt**. Setelah index didapatkan, simpan index tersebut kedalam variabel **resultMtf**. Contoh: huruf pertama dari kata **resultBwt** adalah a, maka cari letak huruf a di dalam **mtf**, letak huruf a adalah 1.
3. Setelah didapatkan letak huruf yang dicari, pindah posisi huruf yang dicari ke bagian depan kata **mtf**, lalu pindah posisi terdepan ke posisi 2 kata **mtf**, posisi 2 ke posisi 3, dst. Contoh : $abn→ a$bn.
4. Lakukan perulangan dari langkah 2 dan 3 sejumlah kata dari **resultBwt**.

Penjelasan variabel yang digunakan: **resultBwt** adalah inputan untuk proses ini, **mtf** digunakan sebagai tampungan dari kata yang diolah didalam proses ini, **resultMtf** digunakan sebagai tampungan dari letak-letak huruf yang dicari dalam variabel **mtf**. Hasil dari cara kerja algoritma MTF, bisa dilihat pada Tabel 1.

JUTEI Edisi Volume.3 No.1 April 2019
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.31.148

TABEL 1
PERUBAHAN RESULTBWT MENJADI RESULTMTF

resultBWT	mtf	resultMTF
a	$abn	97
n	a$bn	3
n	na$b	0
b	na$b	3
\0	bna$	3
a	$bna	3
a	a$bn	0

C. Move-to-Front 1 (MTF-1)

MTF-1 merupakan bentuk pengembangan dari algoritma MTF [5]. Cara kerja dari algoritma MTF-1 berdasarkan inputan yang berasal dari variabel **resultBwt**:

1. Simpan rangkaian simbol/huruf dari ASCII 0 - 127 pada variabel **mtf1**.
2. Lakukan pencarian index dalam variabel **mtf1**, dari huruf-huruf yang berasal dari **resultBwt**. Setelah index didapatkan, simpan index tersebut kedalam variabel **resultMtf1**. Contoh : huruf pertama dari kata **resultBwt** adalah a, maka cari letak huruf a di dalam **mtf1**, letak huruf a adalah 1.
3. Setelah didapatkan letak huruf yang dicari, pindah posisi huruf yang dicari ke posisi 2 kata **mtf1**, lalu pindah posisi 2 ke posisi 3, posisi 3 ke posisi 4, dst. Apabila langkah selanjutnya mencari huruf yang sama dengan yang dicari sebelumnya maka pindah dari posisi 2 ke posisi 1, posisi 1 ke posisi 2. Contoh : cari n dari a$bn → an$b, cari n → na$b.
4. Lakukan perulangan dari langkah 2 dan 3 sejumlah karakter dari **resultBwt**.

Contoh algoritma ini bisa dilihat pada Tabel 2 pada kolom **resultBwt, mtf1,** dan **resultMtf1.**

TABEL 2
PERUBAHAN RESULTBWT MENJADI RESULTMTF1 DAN RESULTMTF2

resultBWT	mtf1	resultMtf1	mtf2	resultMtf2
a	$abn	1	$abn	1
n	a$bn	3	$abn	3
n	an$b	1	$nab	1
b	na$b	3	$nab	3
\0	nba$	3	$bna	0
a	n$ba	3	$bna	3
a	na$b	1	$abn	1

D. Move-to-Front 2 (MTF-2)

MTF-2 juga merupakan bentuk pengembangan dari algoritma MTF-1. Cara kerja dari algoritma MTF-2:

1. Simpan rangkaian simbol/huruf dari ASCII 0 - 127 pada variabel **mtf2**.
2. Lakukan pencarian index dalam variabel **mtf2**, dari huruf-huruf yang berasal dari **resultBwt**. Setelah index didapatkan, simpan index tersebut kedalam variabel **resultMtf2**. Contoh : huruf pertama dari kata **resultBwt** adalah a, maka cari letak huruf a didalam **mtf2**, letak huruf a adalah 1.
3. Setelah didapatkan letak huruf yang dicari, pindah posisi huruf yang dicari ke posisi 2 kata **mtf2**, lalu pindah posisi 2 ke posisi 3, posisi 3 ke posisi 4, dst. Untuk berpindah ke posisi 1, huruf yang dicari sebelumnya harus berada di posisi 1, dan huruf yang dicari pada saat ini berada pada posisi 2, setelah itu tukar posisi 1 dan 2 Contoh : cari a dari a$bn → a$bn, cari $ → $abn.
4. Lakukan perulangan dari langkah 2 dan 3 sejumlah kata dari **resultBwt**.

Sebagai penjelasan variabel yang digunakan, **resultBwt** adalah inputan untuk proses ini, **mtf1** digunakan sebagai tampungan dari kata yang diolah didalam proses MTF-1, **resultMtf1** digunakan sebagai tampungan dari letak-letak huruf yang dicari dalam variabel **mtf1**, **mtf2** digunakan sebagai tampungan dari kata yang diolah didalam proses MTF-2, **resultMtf2** digunakan sebagai tampungan dari letak-letak huruf yang dicari dalam variabel **mtf2**. Hasil dari cara kerja algoritma MTF-2, bisa dilihat pada Tabel 2 pada kolom **resultBwt**, **mtf2**, **resultMtf2**.

E. Run Length Encoding 0 (RLE-0)

Menurut Schiller [7], angka 0 adalah indeks yang paling mendominasi rangkaian indeks **resultMtf, resultMtf1** atau **resultMtf2**. Dalam suatu kasus ketika kita menerapkan hasil dari **resultMtf**, **resultMtf1**, atau **resultMtf2** yang kemudian dilanjutkan dengan RLE-0, probabilitas kemunculan angka 0 bisa mencapai 90%, dengan rata-rata kemunculan 60%. Dengan fakta seperti itu maka Burrows dan Wheeler menyarankan untuk menangani angka 0 tersebut secara khusus, yaitu dengan RLE-0. Tabel 3 adalah tabel perubahan jumlah angka 0 dengan kode RLE-0nya.

TABEL 3
KONVERSI JUMLAH ANGKA 0 YANG BERURUTAN

Jumlah Angka 0	RLE-0 Code
1	0
2	1
3	00
4	01
5	10
6	11

Cara transformasi jumlah angka 0 menjadi RLE-0 Code adalah sebagai berikut:

1. Lakukan rekursi untuk merubah jumlah angka 0 menjadi RLE-0 code, dengan menentukan batas bawah dan batas atas jumlah angka 0 terlebih dahulu. Batas bawah dan atas tersebut nantinya akan berguna dalam proses rekursi.
2. Pada awal algoritma RLE-0 batas bawah = 1, dan batas atas = 2, pangkat = 1.
3. Jika jumlah angka 0 tidak diantara batas bawah dan atas, lakukan langkah 4.
4. Batas bawah += $2^{pangkat}$, dan batas atas = batas bawah + $2^{++pangkat}$ - 1. Lakukan langkah ini, sampai jumlah angka 0 berada di antara batas bawah dan atas.
5. Jika jumlah angka 0 sudah berada diantara batas bawah dan atas, lakukan proses rekursi. Setiap awal rekursi, tentukan batas tengah dari batas bawah dan atas pada saat pemanggilan rekursif tersebut.
6. Jika jumlah angka <= batas tengah, maka simpan "0" pada variabel string. Lalu cek apakah batas bawah dan atas memiliki selisih 1. Jika ya maka masuk dalam *base case*, jika tidak maka masuk dalam *recursive case* dengan mengganti batas atas dengan batas tengah pada rekursif selanjutnya.

7. Jika jumlah angka > batas tengah, maka simpan "1" pada variabel string. Lalu cek apakah batas bawah dan atas memiliki selisih 1. Jika ya maka masuk dalam *base case*, jika tidak maka masuk dalam *recursive case* dengan mengganti batas bawah dengan batas tengah+1 pada rekursi selanjutnya.

Contoh: Semisal jumlah angka 0 adalah 4, batas bawah 1, batas atas 2, pangkat = 1.

- 4 tidak berada di antara batas bawah dan batas atas. Batas bawah = $1+2^1 = 3$, batas atas = $3+2^2-1 = 6$, pangkat=2.
- 4 berada di antara batas bawah dan batas atas, masuk ke dalam proses rekursi.
- Batas tengah = $(3+6)/2 \approx 4$. Oleh karena $4 \leq 4$ maka simpan "0" pada string, lalu masuk pada *recursive case* dengan batas bawah = 3 dan batas atas = batas tengah = 4.
- Batas tengah = $(3+4)/2 \approx 3$. Oleh karena $3 < 4$ maka simpan "1" pada string, lalu masuk pada *base case* karena batas bawah (3) dan batas atas (4) memiliki selisih jumlah 1.
- Hasil transformasi angka 0 menjadi RLE code pada kasus ini adalah 01.

Lalu angka lain hasil dari proses MTF sebelumnya yang bukan angka 0 tidak akan diproses oleh proses RLE, dengan catatan angka di atas 10 atau lebih dari 2 digit harus ditambahkan spasi di awal dan akhir angka tersebut.

F. Run Length Decoding 0

Cara kerja dekompresi dari RLE-0 adalah mengubah rangkaian 0 dan 1 yang khusus pada inputan dengan rangkaian 0:

1. Cari RLE-0 code dengan cara mencari simbol '1' (penulis menyisipkan simbol '!' di bagian depan dan belakang untuk membedakan RLE-0 code dengan yang lain).
2. Hitung berapa banyak jumlah angka 0 dan 1 didalam RLE-0 code tersebut.
3. Setelah mengetahui jumlah RLE-0 code, cari batas bawah dan atas dengan melakukan perulangan sebanyak jumlah RLE-0 code yang telah diketahui.
4. Sebelum melakukan perulangan, nilai batas bawah dan atas = 0 dan pangkat = 0. Setiap melakukan perulangan, batas bawah += $2^{pangkat}$, batas atas = batas bawah + $2^{++pangkat}$ - 1.
5. Setelah mendapatkan batas bawah dan atas, lakukan perulangan sebanyak jumlah RLE-0 code yang telah diketahui untuk menentukan jumlah angka 0.
6. Pada setiap perulangan, cek apakah variabel i sudah mencapai akhir string RLE-0 atau belum.
7. Jika belum, cek apakah RLE-0 pada indeks ke-i adalah 0 atau 1. Jika 0 maka ubah batas atas menjadi (batas bawah + atas) / 2. Jika 1 maka ubah batas bawah menjadi (batas bawah + atas) / 2 + 1.
8. Jika sudah mencapai akhir string RLE-0, cek apakah RLE-0 pada indeks terakhir adalah 0 atau 1. Jika 0 maka jumlah angka 0 = batas bawah, jika 1 maka jumlah angka 0 = batas atas.

Contoh: RLE-0 code = 11.

- Jika RLE-0 code = 11, maka jumlah angka 0 dan 1 di dalam RLE-0 code tersebut adalah 2.
- Lakukan perulangan sebanyak 2 kali. Pada perulangan pertama, batas bawah = $0 + 2^0 = 1$, batas atas = $1 + 2^1 - 1 = 2$. Pada perulangan kedua didapat batas bawah = 3 dan batas atas = 6.
- Lakukan perulangan lagi sebanyak 2 kali untuk mengubah RLE-0 code menjadi rangkaian angka 0.
- Perulangan pertama belum mencapai akhir string RLE-0, string RLE-0 pada indeks ke-0 berupa 1, maka ubah batas bawah menjadi $(3+6)/2 \approx 4$.
- Perulangan kedua sudah mencapai akhir string RLE-0, string RLE-0 pada indeks ke-1/akhir berupa 1, berarti jumlah angka 0 sama dengan batas atas = 6.

G. Inverse MTF

Cara kerja dari algoritma inverse MTF dengan inputan **reverseRle**:

1. Lakukan pencarian karakter dalam variabel **mtf** dengan index dari **reverseRle**. Simpan karakter yang telah dicari kedalam variabel **reverseMtf**. Contoh : indeks pertama dari **reverseRle** adalah 1, maka cari huruf pada indeks ke-1 pada variabel **mtf**, huruf tersebut adalah a.
2. Setelah didapatkan huruf yang dicari, pindah posisi huruf yang dicari ke bagian depan kata **mtf**, lalu pindah posisi terdepan ke posisi 2 kata **mtf**, posisi 2 ke posisi 3, dst. Contoh : \$abn → a\$bn.
3. Lakukan perulangan dari langkah 1 dan 2 sejumlah indeks dari **reverseRle**.

Penjelasan variabel yang digunakan, **reverseRle** adalah inputan untuk proses ini, **mtf** digunakan sebagai tampungan dari kata yang telah disimpan pada saat proses kompresi, **reverseMtf** digunakan sebagai tampungan dari huruf-huruf yang dicari dalam variabel **mtf**. Hasil dari cara kerja algoritma Inverse MTF bisa dilihat pada Tabel 4 berikut.

TABEL 4
PERUBAHAN REVERSERLE MENJADI REVERSEMTF

reverseRle	mtf	reverseMtf
1	\$abn	a
3	a\$bn	n
0	na\$b	n
3	na\$b	b
3	bna\$	\$
3	\$bna	a
0	a\$bn	a

H. Inverse MTF-1

Cara kerja dari algoritma inverse MTF-1 dengan inputan **reverseRle1**:

1. Lakukan pencarian karakter dalam variabel **mtf1** dengan indeks dari **reverseRle1**. Simpan karakter yang telah dicari kedalam variabel **reverseMtf1.** Contoh: indeks pertama dari **reverseRle1** adalah 1, maka cari huruf pada index ke-1 pada variabel **mtf1**, huruf tersebut adalah a.
2. Setelah didapatkan huruf yang dicari, pindah posisi huruf yang dicari ke posisi 2 kata **mtf1**, lalu pindah posisi 2 ke posisi 3, posisi 3 ke posisi 4, dst. Apabila langkah selanjutnya mencari huruf yang sama dengan yang dicari sebelumnya maka pindah dari posisi 2 ke posisi 1, posisi 1 pindah ke posisi 2, posisi 2 ke posisi 3, dst. Contoh: cari n dari a\$bn → an\$b, cari n → na\$b.

JUTEI Edisi Volume.3 No.1 April 2019
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.31.148

3. Lakukan perulangan dari langkah 1 dan 2 sejumlah indeks dari **reverseRle1**.

Hasil dari cara kerja algoritma ini bisa dilihat pada Tabel 5 berikut.

TABEL 5
PERUBAHAN INVERSE MTF-1

reverseRle1	mtf1	reverseMtf1
1	$abn	a
3	a$bn	n
1	an$b	n
3	na$b	b
3	nba$	$
3	n$ba	a
1	na$b	a

I. Inverse MTF-2

Cara kerja dari algoritma inverse MTF-2 dengan inputan **reverseRle2**:

1. Lakukan pencarian karakter dalam variabel **mtf1** dengan indeks dari **reverseRle2**. Simpan karakter yang telah dicari kedalam variabel **reverseMtf2.** Contoh: indeks pertama dari **reverseRle2** adalah 1, maka cari huruf pada index ke-1 pada variabel **mtf2**, huruf tersebut adalah a.
2. Setelah didapatkan huruf yang dicari, pindah posisi huruf yang dicari ke posisi 2 kata **mtf2**, lalu pindah posisi 2 ke posisi 3. Pada MTF-2 untuk berpindah ke posisi pertama hanya bisa dilakukan apabila langkah sebelumnya membuat posisi huruf yang dicari berada di posisi 1 dan huruf yang dicari saat ini ada di posisi 2. Contoh: cari a dari a$bn → a$bn, cari $ → $abn.
3. Lakukan perulangan dari langkah 1 dan 2 sejumlah indeks dari **reverseRle2**.

Hasil dari cara kerja algoritma ini bisa dilihat pada Tabel 6 berikut.

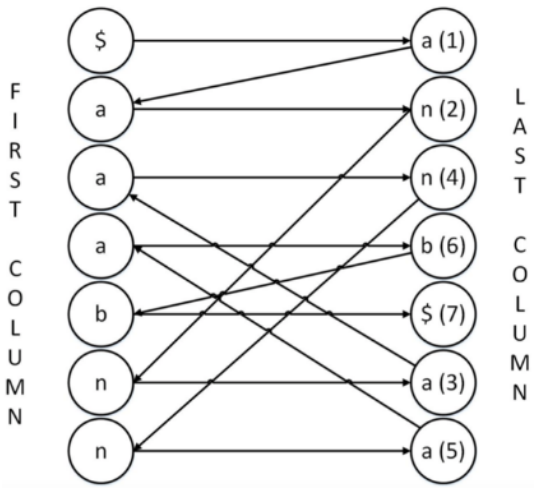
TABEL 6
PERUBAHAN INVERSE MTF-2

reverseRle1	mtf1	reverseMtf1
1	$abn	a
3	$abn	n
1	$nab	n
3	$nab	b
0	$bna	$
3	$bna	a
1	$abn	a

J. Inverse BWT

Setelah Inverse MTF, MTF-1 atau MTF-2 berhasil dilakukan, tahap berikutnya memasukkan hasilnya ke proses Inverse BWT melalui pendekatan Schiller [6]. Inverse bisa dilakukan ketika sistem mengetahui *first column* dan *last column* dari matriks BWT. Penulis telah menyimpan *first column* ke dalam variabel **bwtReverse** dan *last column* didapatkan dari **reverseMtf, reverseMtf1,** atau **reverseMtf2**. Setelah mendapatkan *first column* dan *last column*, lakukan Inverse BWT dengan cara:

1. Mulai proses inverse melalui baris pertama dalam *first column*. Pindah ke bagian *last column* pada baris pertama. Perhatikan Gambar 1.

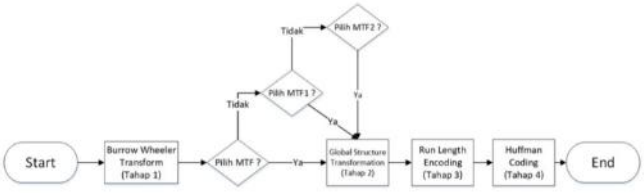


Gambar 1. Langkah pengerjaan Inverse BWT.

2. Sistem mengambil nilai dari *last column* beserta posisi dari nilai tersebut berdasarkan simbolnya, lalu mencari pada *first column* baris keberapa yang mempunyai nilai dan posisi yang sama. Simpan nilai dari *last column* ke dalam variabel original. Contoh: Nilai dari *last column* baris pertama adalah a posisi pertama, maka langkah selanjutnya adalah sistem harus mencari a posisi pertama di *first column* yaitu pada baris 2.
3. Lalu pindah ke bagian *last column* pada baris yang sama.
4. Lakukan langkah 2 dan 3 sebanyak huruf *first column*/ *last column*.
5. Setelah langkah 4 selesai, kita akan mendapatkan kata yang semula tetapi masih terbalik. Maka lakukan pembalikan kata dari variabel **original**. Contoh: ananab$ → $banana.
6. Langkah terakhir, hilangkan simbol *sentinel*/$ pada variabel **original**. $banana → banana.

III. METODOLOGI PENELITIAN

Penelitian dilakukan dengan cara pembuatan program kompresi dengan susunan algoritma yang sudah dibahas sebelumnya, diikuti dengan pengujian menggunakan data uji Alkitab dan Calgary Corpus. Proses kompresi digambarkan dalam diagram alur pada Gambar 2.



Gambar 2. Proses kompresi pada sistem.

Penjelasan tiap langkah yang dilalui pada proses kompresi adalah sebagai berikut:

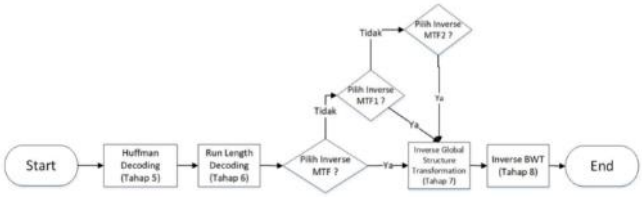
1. BWT
   a. Penambahan Karakter dengan ASCII terkecil pada akhir data.
   b. Rotasi sebanyak panjang data.
   c. Urutkan berdasarkan ASCII dari kecil ke besar.
   d. Ambil karakter terakhir pada tiap rotasi.
2. GST
   a. Pilih salah satu diantara ketiga algoritma yang dibandingkan, yaitu MTF, MTF-1 atau MTF-2.

b. Jika memilih MTF, perpindahan yang dilakukan algoritma ini selalu pada posisi awal array dengan menyimpan posisi karakter yang berpindah.
c. Jika memilih MTF-1, perpindahan yang dilakukan algoritma ini pada posisi awal jika karakter yang berpindah berada di posisi 0 & 1 pada array, jika tidak karakter hanya boleh berpindah ke posisi 1 array.
d. Jika memilih MTF-2, perpindahan yang dilakukan algoritma ini pada posisi awal jika karakter yang berpindah sebelumnya berada di posisi awal dan posisi karakter yang berpindah pada saat ini berada di posisi 0 & 1 pada array, jika tidak karakter hanya boleh berpindah ke posisi 1 array.
e. Algoritma ini menyimpan posisi karakter yang berpindah.

3. RLE
a. Pada tiap posisi yang dihasilkan oleh Global Structure Transformation, cek apakah posisi tersebut 0 atau bukan.
b. Jika bukan tampung posisi tersebut kedalam string.
c. Jika ya, cari total urutan posisi 0 tersebut.
d. Cari batas bawah dan atas total urutan posisi 0.
e. Transformasi urutan 0 tersebut dengan menggunakan total urutan 0, batas bawah, dan batas atas.

4. EC (Huffman Coding)
a. Buat tabel untuk semua simbol, diurutkan berdasarkan frekuensi simbol terlebih dahulu, lalu diurutkan berdasarkan nilai ASCII dari kecil ke besar.
b. Bangun sebuah *tree* dengan cara memilih 2 simbol dengan frekuensi terkecil, satukan kedua simbol tersebut, lalu jumlahkan kedua frekuensi simbol tersebut, lalu masukkan kedalam tabel kembali secara khusus. Lakukan langkah ini secara terus menerus hingga terbentuk 1 rangkaian simbol saja pada tabel.
c. Ubah setiap simbol yang dihasilkan Run Length Encoding menjadi rangkaian bit sesuai dari *tree* yang terbentuk.

Sedangkan proses dekompresi dilakukan sesuai alur pada Gambar 3.



Gambar 3. Proses dekompresi pada sistem.

Penjelasan tiap langkah yang dilalui pada proses dekompresi adalah sebagai berikut:

1. Huffman Decoding
a. Bangun *tree* melalui rangkaian *bit tree* yang ada pada teks kompresi.
b. Telusuri semua rangkaian bit kompresi hingga rangkaian bit terakhir.

2. Run Length Decoding
a. Cek setiap karakter yang dihasilkan oleh Huffman Decoding, apakah karakter tersebut merupakan rangkaian RLE atau bukan.
b. Jika karakter tersebut bukan rangkaian RLE, maka simpan karakter tersebut ke dalam array.
c. Jika karakter tersebut merupakan rangkaian RLE, maka cari batas bawah dan atas rangkaian RLE tersebut.
d. Transformasi rangkaian RLE tersebut menggunakan rangkaian RLE, batas bawah dan atas.

3. Inverse GST
a. Pilih salah satu diantara ketiga algoritma yang dibandingkan, yaitu Inverse MTF, MTF-1, atau MTF2.
b. Jika memilih Inverse MTF, perpindahan yang dilakukan algoritma ini selalu pada posisi awal array.
c. Jika memilih Inverse MTF1, perpindahan yang dilakukan algoritma ini pada posisi awal jika karakter yang berpindah berada di posisi 0 & 1 pada array, jika tidak karakter hanya boleh berpindah ke posisi 1 array.
d. Jika memilih Inverse MTF2, perpindahan yang dilakukan algoritma ini pada posisi awal jika karakter yang berpindah sebelumnya berada di posisi awal dan posisi karakter yang berpindah pada saat ini berada di posisi 0 & 1 pada array, jika tidak karakter hanya boleh berpindah ke posisi 1 array.
e. Algoritma ini menyimpan karakter yang berpindah.

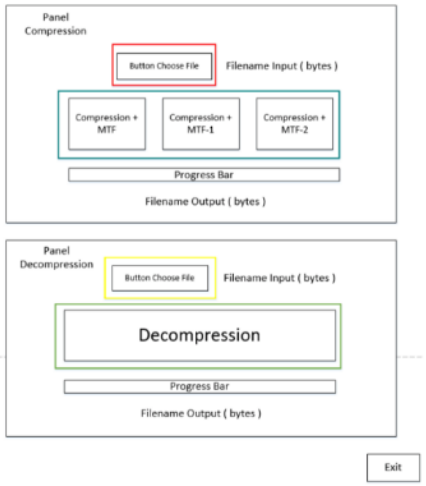
4. Inverse BWT
a. Buat sebuah array yang berisi karakter dan total karakter, hasil dari Inverse GST.
b. Urutkan array tersebut berdasarkan nilai ASCII, dari kecil ke besar.
c. Buat sebuah array yang berisi rank tiap karakter, hasil dari Inverse GST.
d. Lakukan perulangan sejumlah karakter yang dihasilkan Inverse GST.
e. Mulai dari array pertama, simpan karakter array pertama tersebut ke dalam sebuah string.
f. Lakukan perpindahan berdasarkan rank karakter yang berpindah + jumlah total karakter karakter sebelumnya.
g. Lalu simpan karakter array hasil perpindahan ke dalam sebuah string.
h. Lakukan langkah f dan g hingga perulangan selesai.
i. Setelah perulangan selesai cetak string yang menyimpan karakter-karakter yang telah diolah pada proses sebelumnya, dimulai dari 2 karakter terakhir.

Program diimplementasikan menggunakan bahasa Visual C++. Rancangan antarmuka memenuhi ketentuan sebagai berikut:

- Program hanya memiliki 1 form.
- Terdapat 2 panel, yaitu panel kompresi dan panel dekompresi.
- Ketika ingin melakukan kompresi, maka pengguna harus memilih file terlebih dahulu dengan memilih *button* Choose File yang berada di kotak merah pada Gambar 4.
- Setelah memilih file, pilih salah satu dari 3 button yang berada di kotak biru pada Gambar 4, yaitu Compression + MTF, Compression + MTF-1, Compression + MTF-2.
- Ketika ingin melakukan dekompresi, maka pengguna harus memilih file terlebih dahulu dengan memilih *button* Choose File yang berada di kotak kuning pada Gambar 4.

JUTEI Edisi Volume.3 No.1 April 2019
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.31.148

- Setelah memilih file, pilih button Decompression yang berada di kotak hijau pada Gambar 4.



Gambar 4. Rancangan antarmuka program.

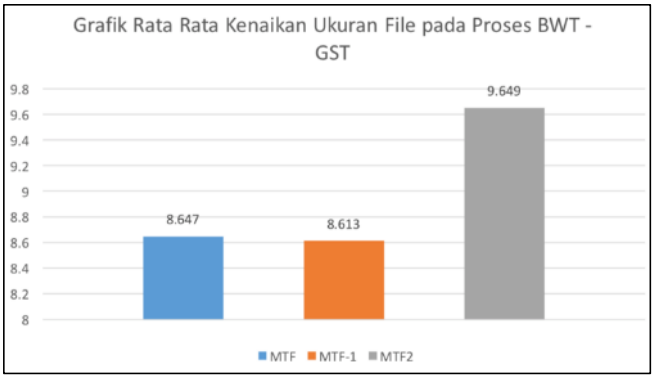
IV. HASIL DAN ANALISIS

Hasil teks yang telah dikompresi kemudian didekompresi kembali dan dibandingkan dengan teks asli menggunakan aplikasi Meld. Jika Meld mendeteksi adanya perbedaan pada teks yang dibandingkan, maka Meld akan menggarisbawahi bagian yang berbeda antar teks, seperti ditunjukkan pada Gambar 5. Jika Meld tidak mendeteksi adanya perbedaan pada teks yang dibandingkan, maka Meld akan mengeluarkan pesan *"File are identical"*.

Gambar 5. Terdapat perbedaan antara kedua teks yang dibandingkan.

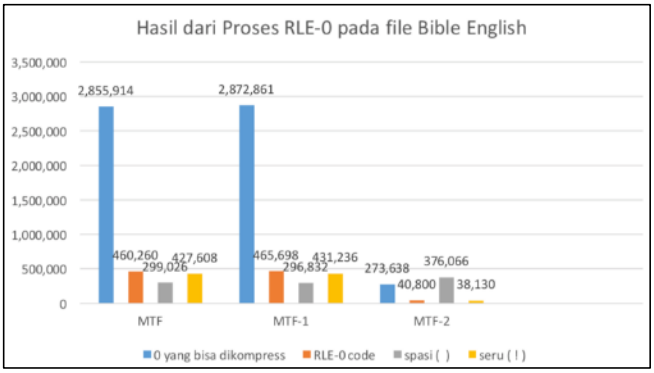
Semua file yang diuji bisa kembali didekompresi 100% seperti file aslinya. Untuk meneliti dan menganalisis dampak perubahan masing-masing proses algoritma terhadap file maka setiap tahapan kompresi diukur hasil sementara yang tercipta oleh sistem.

Ketika dilakukan pengujian pada tahap proses GST (MTF/MTF-1/MTF-2), didapatkan kenaikan ukuran file yang ditunjukkan pada Gambar 6. Kenaikan ukuran diketahui paling sedikit melalui proses MTF-1 dengan kenaikan rata-rata sebesar 8,613%.

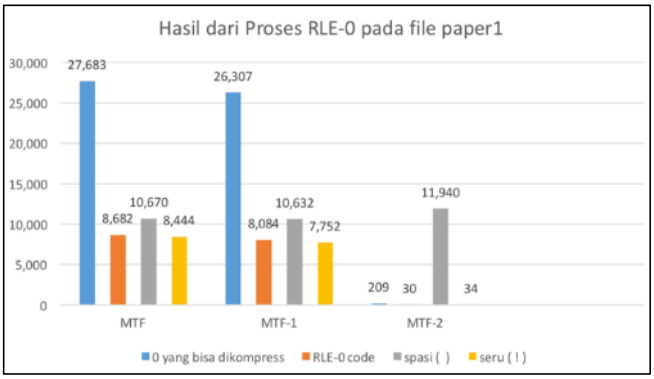


Gambar 6. Grafik rata-rata kenaikan ukuran file pada proses BWT-GST.

Berikutnya, hasil dari masing-masing proses GST dilanjutkan dengan proses RLE-0. Dari hasil pengujian didapatkan bahwa terjadi penurunan ukuran file 8,647% ketika menggunakan MTF dan 7,597% ketika menggunakan MTF-1. Sedangkan penggunaan MTF-2 membuat ukuran file bertambah. Hal ini disebabkan karena total simbol spasi dan seru yang dihasilkan MTF lebih sedikit daripada MTF-1. Terlihat pada Gambar 7 dan 8 sebagai proses RLE-0 pada file Bible English dan Calgary Corpus paper1.

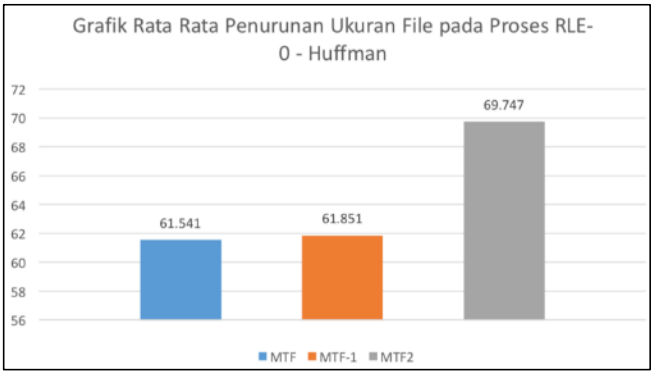


Gambar 7. Hasil dari Proses RLE-0 pada file Bible English.



Gambar 8. Hasil dari Proses RLE-0 pada file Calgary Corpus paper1.

Langkah berikutnya, hasil proses RLE-0 dilanjutkan dengan proses Entropy Coding menggunakan algoritma Huffman. Dari hasil pengujian diketahui bahwa rata-rata penurunan ukuran file paling besar didapat dari proses MTF-2 dengan nilai 69,747%. Hal ini disebabkan MTF-2 cenderung membuat total bit tiap simbol lebih sedikit dibanding dua proses yang lain. Perhatikan Gambar 9.



Gambar 9. Rata-rata penurunan ukuran file pada proses Entropy Coding.

Setelah Entropy Coding selesai, maka file kompresi

dihitung rasio kompresinya dengan persamaan (1) [8].

$$CR = \frac{output\ file}{input\ file} \times 100\% \quad (1)$$

Hasil pengujian dari setiap data uji disajikan pada Tabel 7, Tabel 8 dan Tabel 9. Dari ketiga tabel tersebut diketahui hasil dari MTF-1 memiliki rasio kompresi yang terbaik, disusul oleh MTF-2 dan yang paling buruk MTF.

JUTEI Edisi Volume.3 No.1 April 2019
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.31.148

TABEL 7
HASIL KOMPRESI MENGGUNAKAN GST MTF

File	Ukuran Asli	Ukuran Setelah GST	Ukuran Setelah RLE-0	Ukuran Setelah Huffman	Rasio Kompresi
	(bytes)				
Bible (English)	4.467.663	4.617.177	2.948.157	1.129.191	25,275%
Bible (Indonesian)	5.942.709	6.145.542	4.175.986	1.567.111	26,370%
Bible (Javanese)	5.321.906	5.516.529	4.826.109	1.595.219	29,975%
bib	111.261	120.388	98.047	38.103	34,247%
book	610.856	651.374	560.993	215.438	35,268%
news	377.109	420.892	421.254	164.782	43,696%
paper1	53.161	58.497	58.610	22.810	42,907%
paper2	82.199	89.348	87.161	34.095	41,479%
paper3	46.526	51.566	55.085	22.835	49,080%
paper4	13.286	15.098	17.512	6.970	52,461%
paper5	11.954	13.820	16.470	6.578	55,028%
paper6	38.105	42.043	43.194	16.864	44,257%
progc	39.611	44.221	45.061	17.375	43,864%
progl	71.646	75.994	59.139	22.175	30,951%
propgp	49.379	52.389	40.727	15.186	30,754%
				Mean Rasio	**39,041%**

TABEL 8
HASIL KOMPRESI MENGGUNAKAN GST MTF-1

File	Ukuran Asli	Ukuran Setelah GST	Ukuran Setelah RLE-0	Ukuran Setelah Huffman	Rasio Kompresi
	(bytes)				
Bible (English)	4.467.663	4.616.080	2.936.985	1.118.665	25,039%
Bible (Indonesian)	5.942.709	6.144.318	4.158.902	1.550.535	26,091%
Bible (Javanese)	5.321.906	5.515.116	4.799.209	1.581.410	29,715%
bib	111.261	120.386	98.236	37.736	33,917%
book	610.856	651.124	559.987	213.849	35,008%
news	377.109	420.691	419.973	162.393	43,063%
paper1	53.161	58.478	58.639	22.581	42,477%
paper2	82.199	89.326	87.121	33.848	41,178%
paper3	46.526	51.515	54.950	22.761	48,921%
paper4	13.286	15.082	17.418	6.873	51,731%
paper5	11.954	13.812	16.450	6.504	54,409%
paper6	38.105	42.041	43.326	16.720	43,879%
progc	39.611	44.194	45.110	17.228	43,493%
progl	71.646	75.999	59.597	22.042	30,765%
propgp	49.379	52.386	40.685	15.092	30,564%
				Mean Rasio	**38,683%**

TABEL 9
HASIL KOMPRESI MENGGUNAKAN GST MTF-2

File	Ukuran Asli	Ukuran Setelah GST	Ukuran Setelah RLE-0	Ukuran Setelah Huffman	Rasio Kompresi
	(bytes)				
Bible (English)	4.467.663	4.655.697	4.870.047	1.280.605	28,664%
Bible (Indonesian)	5.942.709	6.191.755	6.689.845	1.687.041	28,388%
Bible (Javanese)	5.321.906	5.559.540	5.789.824	1.648.255	30,971%
bib	111.261	121.662	136.923	38.633	34,723%
book	610.856	656.005	725.547	220.429	36,085%
news	377.109	425.231	520.563	161.958	42,947%
paper1	53.161	59.132	70.927	21.643	40,712%
paper2	82.199	90.341	105.407	33.594	40,869%
paper3	46.526	52.232	63.535	25.414	54,623%
paper4	13.286	15.317	19.377	6.584	49,556%
paper5	11.954	13.952	17.790	6.269	52,443%
paper6	38.105	42.412	50.484	16.066	42,162%
progc	39.611	44.482	53.630	16.670	42,084%
progl	71.646	76.457	86.077	21.428	29,908%
propgp	49.379	52.674	59.262	14.633	29,634%
				Mean Rasio	**38,918%**

Dari ketiga mean rasio kompresi terlihat bahwa MTF-1 memberikan rata-rata rasio kompresi yang terkecil. Semakin kecil nilai rasio kompresi, hasil kompresi dikatakan semakin baik [9].

Jika dianalisis berdasarkan bahasa, kita bisa mencermati obyek uji Bible dalam English, Indonesian dan Javanese. Untuk English Bible terjadi kenaikan ukuran rata-rata sebesar 103,63% setelah diproses GST, kemudian menurun dengan kisaran 80,24% setelah dilakukan RLE, dan hasil akhir rasionya menjadi 26,326%. Sedangkan Indonesian Bible, kenaikan ukuran setelah proses GST sebesar 103,67%, kemudian menurun sebesar 84,28% setelah RLE dan hasil akhirnya memiliki rasio kompresi 26,95%. Kemudian untuk Javanese Bible terjadi kenaikan ukuran sebesar 103,92% setelah pemrosesan GST, lalu menjadi 96,55% setelah pemrosesan RLE. Hasil akhir didapat rasio 30,220% yang berarti merupakan rasio terburuk dari ketiga bahan uji. Semakin kecil rasio, hasil kompresi dikatakan semakin baik.

Ditinjau dari besar ukuran file asli, rasio kompresi hasil akhir kurang menunjukkan pola yang kentara. Seperti pada kompresi teks *lossless* pada umumnya, hasil kompresi akan memberikan rasio yang baik jika konten teks memiliki karakter yang berulang cukup banyak.

V. KESIMPULAN

Berdasarkan hasil pengujian dan analisis sistem yang telah dilakukan, maka dapat disimpulkan bahwa:

1. MTF-1 merupakan algoritma GST yang mampu memberikan rasio kompresi yang terbaik dibandingkan oleh MTF dan MTF-2 untuk data Alkitab berbahasa Inggris, Indonesia, Jawa dikarenakan jumlah total tiap bit pada proses Huffman lebih sedikit dibandingkan 2 proses lainnya.
2. Pada data Calgary Corpus (bib, paper1, paper5, paper6, progc, progl) memiliki ukuran yang lebih kecil ketika melalui proses MTF dibandingkan MTF-1 dan MTF-2, sedangkan sisa data Calgary Corpus yang lain (book, news, paper2, paper3, paper4, progp) memiliki ukuran yang lebih kecil ketika melalui proses MTF-1 dibandingkan dengan MTF dan MTF-2 dikarenakan jumlah total tiap bit pada proses Huffman lebih sedikit dibandingkan 2 proses lainnya.
3. Hasil akhir BWCA dihasilkan oleh proses Huffman, dimana proses tersebut sangat bergantung pada total bit tiap simbol, sedangkan proses RLE-0 bergantung dari seberapa banyak 0 yang bisa dikompres yang dihasilkan pada proses GST (MTF, MTF-1, dan MTF-2), dan RLE-0 code, simbol spasi, dan seru yang dihasilkan pada proses RLE-0.
4. Untuk pengujian bahasa pada Bible, didapatkan hasil terbaik untuk Bible dalam bahasa Inggris (rasio 26,326%), dan hasil terburuk untuk Bible dalam bahasa Jawa (rasio 30,22%).

Adapun saran untuk perbaikan, perkembangan dan *future works* adalah memperbaiki kompleksitas algoritma dari O(n log n^2) menjadi O(n log n). Algoritma ini juga tidak menutup kemungkinan diterapkan pada kompresi

JUTEI Edisi Volume.3 No.1 April 2019
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2018.31.148

citra/*image* dengan menggantikan karakter dengan nilai warna tiap pixelnya, atau untuk kompresi lossless audio [10].

UCAPAN TERIMA KASIH

Dalam menyelesaikan penelitian ini, penulis mengucapkan terima kasih kepada Bapak Lukas Chrisantyo dan Bapak Yuan Lukito yang telah mengarahkan penelitian dengan baik. Juga diucapkan terima kasih kepada keluarga dan teman-teman seperjuangan program studi Informatika UKDW angkatan 2013 yang selalu setia mendukung dan mendoakan setiap saat.

DAFTAR PUSTAKA

[1] K. Sayood. (2017). *Introduction to Data Compression*. San Fransisco: Elsevier.

[2] R.R. Baruah & M.P. Bhuyan. (2014). Enhancing Dictionary Based Preprocessing for Better Text Compression. *International Journal of Computer Trends and Technology*. *9(1)*, hal. 4-9.

[3] A. Andersson & S. Nilsson. (1994). A New Efficient Radix Sort. *SFCS '94 Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 714-721.

[4] H. Itoh & H. Tanaka. (2003). An Efficient Method in Memory Construction of Suffix Arrays. *SPIRE'99 Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, 81.

[5] S.S. Nanda, K. Das, J. Padhi & S. Hota, "Advanced Move-to-Front for List Access Problem" dipresentasikan di *International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, India, 18-19 Maret 2016.

[6] S. Deorowicz. (2003). *Universal lossless data compression algorithms*. Czech: Silesian University of Technology, Faculty of Automatic Control, Electronics and Computer Science, Institute of Computer Science.

[7] D. Schiller. (2012). The Burrows-Wheeler Algorithm. Theoretical Computer Science Department, RWTH Aachen University, Germany. [Online]. Tersedia: htttp://tcs.rwth-aachen.de/lehre/Komprimierung/SS2012/ausarbeitungen/Burrows-Wheeler.pdf.

[8] D. Solomon. (2010). *Handbook of Data Compression.* California: Springer.

[9] C. McAnlis & A. Haecky. (2016). *Understanding Compression: Data Compression for Modern Developers*. California: O'Reilly Media.

[10] H.A. Elsayed, "Burrows-Wheeler Transform and combination of Move-to-Front coding and Run Length Encoding for lossless audio coding" dipresentasikan di *The 9th International Conference on Computer Engineering & Systems (ICCES)*, Mesir, 22-23 Desember 2014.