Studi Kompresi Data dengan Metode *Arithmetic Coding*

Petrus Santoso
Fakultas Teknologi Industri, Jurusan Teknik Elektro, Universitas Kristen Petra
e-mail: P.Santoso@cs.utwente.nl

Abstrak

Ada banyak sekali metode kompresi data yang ada saat ini. Sebagian besar metode tersebut bisa dikelompokkan ke dalam salah satu dari dua kelompok besar, *statistical based* dan *dictionary based*. Contoh dari *dictionary based coding* adalah Lempel Ziv Welch dan contoh dari *statistical based* coding adalah *Huffman Coding* dan *Arithmetic Coding* yang merupakan algoritma terbaru. Makalah ini mengulas prinsip-prinsip dari Arithmetic Coding serta keuntungan-keuntungannya dibandingkan dengan metode Huffman Coding. Permasalahan pada implementasi *arithmetic coding* karena keterbatasan pengolahan floating point pada encoder maupun decoder, juga dibahas dengan usulan solusi berupa modifikasi arithmatic coding dengan menggunakan bilangan integer. Pada akhirnya ditarik kesimpulan, bahwa algoritma ini cukup baik untuk dipakai dalam keperluan kompresi data. Alasan pertama karena jumlah *coding bit* pada *arithmatic coding* lebih sedikit dibandingkan dengan Huffman Coding. Modifikasi dengan menggunakan bilangan integer juga mampu mengatasi keterbatasan peralatan-peralatan encoder dan decoder dari pengolahan *floating ponit* yang terlalu panjang. Kedua karena jumlah bit kodenya lebih sedikit dan dapat diimplementasikan.

Kata kunci : *Arithmetic Coding*, kompresi data, algoritma

Abstract

Nowadays, there are many methods of data compression available.Most of them can be classified into one of the two big categories, i.e., statistical based and dictionary based. An Example of dictionary based coding is Lempel Zip Welch. The example of statistical based coding are Huffman coding and Arithmetic coding, as the newest algorithm. This paper describes the principles of Arithmetic coding along with its advantages compared to Huffman coding method. Problems of arithmetic coding implementation, due to the restriction of the floating point processing either on encoder nor decoder, are discussed as well with the suggested solution, i.e.modification of arithmetic coding using numerical integer. As the final conclusion, the algorithm is outstanding for the use of data compression matters. The number of bit coding of arithmetic is less than that of Huffman coding. The modification with the numerical integer is capable of dealing with the limitations of the encoder and decoder equipments over to long floating point processing. Due to the less number of the bit codings, and can be implemented.

Keywords: *Arithmetic Coding, data compression, algorithm*

Pendahuluan

Metode kompresi data bisa dibagi menjadi dua kelompok besar: *dictionary based* dan *statistical based*. *Dictionary based* bekerja dengan mengganti kelompok simbol dalam data menjadi kode-kode dengan panjang tertentu. Dengan asumsi kode-kode tersebut secara umum lebih pendek dari kelompok simbol yang digantikan. *Statistical based* melakukan kompresi dengan pendekatan yang berbeda. Simbol-simbol yang ada di-*encode* satu per satu. Panjang dari kode output akan bervariasi tergantung dari probabilitas / frekuensi pemunculan simbol.

Catatan : Diskusi untuk makalah ini diterima sebelum tanggal 1 Mei 2001. Diskusi yang layak muat akan diterbitkan pada Jurnal Teknik Elektro volume 1 nomor 2 September 2001

Simbol dengan probabilitas rendah akan di-*encode* dengan jumlah bit yang banyak. Sedang simbol dengan probabilitas tinggi akan di-*encode* dengan jumlah bit yang sedikit.

Contoh dari *dictionary based* di antaranya adalah *Lempel Ziv Welch.* Sedangkan contoh dari *statistical based* adalah *Huffman Coding* dan *Arithmetic Coding*. Dalam hal ini algoritma yang paling baru adalah *Arithmetic Coding*.

Dengan latar belakang ini, terlihat masih diperlukannya perbandingan-perbandingan untuk menentukan algoritma mana yang sebaiknya dipakai dalam melakukan kompresi data.

Tujuan utama penulisan makalah ini adalah untuk menunjukkan kelebihan *Arithmetic*

Coding dibandingkan dengan algoritma *Huffman Coding* yang sudah terkenal dan banyak dipakai. Tujuan berikutnya adalah memberikan alternatif implementasi algoritma *Arithmetic Coding* untuk mengatasi masalah presisi bit.

Pada makalah ini terutama akan dibahas tentang *Arithmetic Coding.* Pembahasan akan dilakukan mulai dengan penjelasan algoritma dasar *Arithmetic Coding.* Berikutnya akan dilakukan perbandingan dengan algoritma *Huffman Coding*. Kemudian dilanjutkan dengan alternatif implementasi algoritma *Arithmetic Coding.* Di bagian terakhir akan diambil kesimpulan tentang implementasi bagaimana yang sebaiknya dipakai untuk *Arithmetic Coding*.

Algoritma Dasar

Pada umumnya, algorithma kompresi data melakukan penggantian satu atau lebih simbol input dengan kode tertentu. Berbeda dengan cara tersebut, *Arithmetic Coding* menggantikan satu deretan simbol input dengan sebuah bilangan *floating point.* Semakin panjang dan semakin kompleks pesan yang dikodekan, semakin banyak bit yang diperlukan untuk keperluan tersebut.

Output dari *arithmetic coding* ini adalah satu angka yang lebih kecil dari 1 dan lebih besar atau sama dengan 0. Angka ini secara unik dapat di-*decode* sehingga menghasilkan deretan simbol yang dipakai untuk menghasilkan angka tersebut.

Untuk menghasilkan angka output tersebut, tiap simbol yang akan di-*encode* diberi satu set nilai probabilitas. Contoh, andaikan kata **TELEMATIKA** akan di-*encode*. Akan didapatkan tabel probabilitas berikut :

Tabel 1. Contoh Tabel Probabilitas untuk Kata "TELEMATIKA"

Karakter	Probabilitas
A	2/10
E	2/10
I	1/10
K	1/10
L	1/10
M	1/10
T	2/10

Setelah probabilitas tiap karakter diketahui. Tiap simbol/karakter akan diberikan range tertentu yang nilainya berkisar di antara 0 dan 1, sesuai dengan probabilitas yang ada. Dalam hal ini tidak ada ketentuan urut-urutan penentuan segmen, asalkan antara *encoder* dan *decoder* melakukan hal yang sama.

Dari tabel 1 di atas dibentuk tabel 2 berikut:

Tabel 2. Range Simbol untuk Kata "TELEMATIKA"

Karakter	Probabilitas	Range
A	2/10	0.00 – 0.20
E	2/10	0.20 – 0.40
I	1/10	0.40 – 0.50
K	1/10	0.50 – 0.60
L	1/10	0.60 – 0.70
M	1/10	0.70 – 0.80
T	2/10	0.80 – 1.00

Dari tabel ini, satu hal yang perlu dicatat adalah tiap karakter melingkupi range yang disebutkan kecuali bilangan yang tinggi.

Jadi huruf/simbol 'T' sesungguhnya mempunyai range mulai dari 0.80 sampai dengan 0.9999…..

Selanjutnya untuk melakukan proses *encoding* dipakai algoritma berikut:

- Set low = 0.0
- Set high = 1.0
- While (simbol input masih ada) do
  - Ambil simbol input
  - CR = high – low
  - High = low + CR*high_range (simbol)
  - Low = low + CR*low_range (simbol)
- End While
- Cetak Low

Di sini 'Low' adalah output dari proses *arithmetic coding*.

Untuk kata 'TELEMATIKA' di atas, pertama kita ambil karakter 'T'. Nilai CR adalah 1-0 = 1. High_range (T) = 1.00, Low_range(T) = 0.80.

Kemudian didapatkan nilai

high = 0.00 + CR*1.00 =1.00
low = 0.00 + CR*0.80 = 0.80

Kemudian diambil karakter 'E'. Nilai CR adalah 1.0 – 0.8 = 0.2. High_range(E)= 0.40, Low_range (E) = 0.20. Kemudian didapatkan nilai

high = 0.80 + CR*0.4 = 0.88
low = 0.80 + CR*0.2 = 0.84

Dan seterusnya yang diringkaskan dalam tabel 3 berikut.

Tabel 3. Proses Encoding untuk Kata "TELEMATIKA"

S	Low	High	CR
	0.0	1.0	1.0
T	0.8	1.0	0.2
E	0.84	0.88	0.04
L	0.864	0.868	0.004
E	0.8648	0.8656	0.0008
M	0.86536	0.86544	8E-05
A	0.86536	0,86544	1.6E-05
T	0.8653728	0.865376	3.2E-06
I	0.86537408	0.8653744	3.2E-07
K	0.86537424	0.865374272	3.2E-08
A	0.86537424	0.865374246	6.4E-09

Dari proses ini didapatkan nilai

Low = 0.86537424

Nilai inilah yang ditransmisikan untuk membawa pesan 'TELEMATIKA'.

Untuk melakukan *decoding* dipakai algoritma berikut:

- Ambil *encoded-symbol (ES)*
- Do
  - Cari range dari simbol yang melingkupi *ES*
  - Cetak simbol
  - CR = high_range – low_range
  - ES = ES – low_range
  - ES = ES / CR
- Until simbol habis

Dalam hal ini simbol habis bisa ditandai dengan simbol khusus (End of Message misalnya) atau dengan menyertakan panjang pesan waktu dilakukan transmisi.

Untuk pesan yang tadi di-*encode* (ES = 0.86537424) dilakukan proses *decoding* sebagai berikut. Didapatkan range simbol yang melingkupi ES adalah simbol/karakter 'T'.

Low_range= 0.8
High_range=1.0
CR = 1.0-0.8 = 0.2
ES = 0.86537424 – low_range
ES = 0.06537424
ES = 0.06537424 / CR
ES = 0.3268712

Untuk ringkasnya, proses *decoding* bisa dilihat dalam tabel 4 berikut:

Tabel 4. Proses Decoding untuk Kata "TELEMATIKA"

ES	S	L	H	CR
0.86537424	T	0.8	1.0	0.2
0.3268712	E	0.2	0.4	0.2
0.634356	L	0.6	0.7	0.1
0.34356	E	0.2	0.4	0.2
0.7178	M	0.7	0.8	0.1
0.178	A	0.0	0.2	0.2
0.89	T	0.8	1.0	0.2
0.45	I	0.4	0.5	0.1
0.5	K	0.5	0.6	0.1
0	A	0.0	0.2	0.2

Keuntungan Arithmetic Coding dibandingkan dengan Huffman Coding

Pada bagian ini akan dicoba untuk membandingkan antara *Huffman Coding* dan *Arithmetic Coding.*

Huffman Coding akan menghasilkan 1 kode output untuk tiap simbol yang di-*encode.* Panjang paling pendek adalah 1 bit. Jumlah bit paling optimal yang dipakai untuk melakukan *encoding* adalah $\log_2 1/p$. Di mana p adalah probabilitas dari simbol yang diberikan. Contohnya jika probabilitas sebuah simbol 1/64, maka jumlah bit yang dipakai sebagai kode adalah $\log_2 64 = 6$ bit.

Yang menjadi masalah di sini adalah jumlah bit harus merupakan bilangan bulat. Jadi jika probabilitasnya 1/3. Seharusnya jumlah bit optimal adalah 1.6. Tapi harus dibulatkan menjadi 2, sehingga mengakibatkan kompresi yang kurang optimal.

Masalah ini akan terlihat jika ada salah satu simbol yang mempunyai probabilitas tinggi.

Contoh, simbol X mempuyai probabilitas 90%, sedang EOM (end of message) probabilitasnya 10%. Akan ditransmisikan pesan XXXXXXXX.

Pada *Huffman Coding:*

X butuh 0.15 bit → 1 bit
EOM butuh 3.3 bit → 2 bit

Total untuk pesan tersebut butuh 10 bit

Untuk *arithmetic coding*, proses encoding bisa dilihat pada tabel 5 berikut.

Tabel 5. Proses Encoding untuk Kata "XXXXXXXX"

S	Low	High
	0.0	1.0
X	0.0	0.9
X	0.0	0.81
X	0.0	0.729
X	0.0	0.6561
X	0.0	0,59049
X	0.0	0.531441
X	0.0	0.4782969
X	0.0	0.43046721
EOM	0.387420489	0.43046721

Sebagai output, bisa digunakan angka antara low dan high. Semisal 0.41. Untuk itu cuma dibutuhkan 7 bit. Sedangkan dengan *Huffman Coding* dibutuhkan 10 bit. Andaikata ada 100 simbol, 99 di antaranya adalah X dan 1 simbol EOM. *Huffman Coding* akan memerlukan jauh lebih banyak bit dibandingkan dengan *arithmetic coding.*

Perbandingan kedua adalah dari segi implementasi. Untuk *Huffman Coding* dibutuhkan *binary-tree*. Seperti sudah diketahui, *binary-tree* ini adalah proses yang mahal, apalagi jika diterapkan dengan kondisi statistik yang berubah-ubah (*adaptive coding*). Sedangkan *arithmetic coding* hanya memerlukan operasi aritmetik biasa.

Implementasi Arithmetic Coding

Implementasi Arithmetic Coding harus memperhatikan kemampuan encorder dan decoder, yang pada umumnya mempunyai keterbatasan jumlah mantissa (angka dibelakang koma). Hal ini dapat menyebabkan *"error"*/ kesalahan apabila suatu arithmetic coding mempunyai kode dengan floating point yang sangat panjang.

Masalah ini bisa diatasi dengan meng-implementasikan *arithmetic coding* menggunakan bilangan integer (16 atau 32 bit integer). Hal ini juga amat mempercepat proses, karena perhitungan integer jauh lebih cepat dari perhitungan floating point.

Untuk implementasi tersebut, langkahnya adalah sebagai berikut:

- Pertama ditetapkan nilai High dan Low, sesuai dengan panjang register. Untuk 16 bit High = $FFFF; Low=$0000.

- Tetapkan nilai CR
  CR = High – Low +1

- While (simbol input masih ada) do
  - Ambil simbol input
  - High = low + CR*high_range (simbol) - 1
  - Low = low + CR*low_range (simbol)
  - CR = High – Low + 1
  - IF digit pertama sama maka SHIFT OUT digit pertama

- End While
- SHIFT OUT digit pertama jika masih dibutuhkan
- Cetak Low

Sebagai ilustrasi digunakan contoh pada bagian terdahulu. Dalam hal ini digunakan nilai desimal.

HIGH = 999999
LOW = 000000

Untuk jelasnya proses tersebut bisa dilihat pada tabel 6.

Untuk proses *decoding* prosesnya sama dengan proses sebelumnya. Dalam proses ini ada 3 variabel yang penting High, low, dan output. Nilai high dan low mempunyai korespondensi dengan nilai high dan low yang ada pada proses encoding. Dengan membandingkan digit pertama dari high dan low juga menentukan kapan dilakukan shift in dari kode simbol yang ada.

Tabel 6. Contoh Encoding dengan Bilangan Integer untuk Kata 'TELEMATIKA'

Proses	Output	Low	High	CR
Kondisi awal		000000	999999	1000000
T (0.8 – 1.0)		800000	999999	200000
E (0.2 – 0.4)		840000	879999	
Shift Out	.8	400000	799999	400000
L (0.6 – 0.7)	.8	640000	679999	
Shift Out	.86	400000	799999	400000
E (0.2 – 0.4)	.86	480000	559999	80000
M (0.7 – 0.8)	.86	536000	543999	
Shift Out	.865	360000	439999	80000
A (0.0 – 0.2)	.865	360000	375999	
Shift Out	.8653	600000	759999	160000
T (0.8 – 1.0)	.8653	728000	759999	
Shift Out	.86537	280000	599999	320000
I (0.4 – 0.5)	.86537	408000	439999	
Shift Out	.865374	80000	399999	320000
K (0.5 – 0.6)	.865374	240000	271999	
Shift Out	.8653742	400000	721999	320000
A (0.0 – 0.2)	.8653742	400000	464399	
Shift Out	.86537424	000000	643999	644000
→	.86537424			

Kesimpulan

Dari pembahasan di atas dapat ditarik kesimpulan sebagai berikut :

1. Algoritma *Arithmetic Coding* ini cukup baik dipakai untuk kompresi data
2. Algoritma ini akan lebih optimal dari *Huffman Coding* jika ada data/simbol yang mempunyai probailitas besar (sebagai konsekuensi, jumlah banyak).
3. Untuk implementasi *Arithmetic Coding* sebaiknya dilakukan dengan perhitungan bilangan Integer.

Daftar Pustaka

[1]. Nelson, Mark. *Arithmetic Coding + Statistical Modeling = Data Compression*; Dr. Dobb's Journal, February 1991

[2]. Press, William H. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, 1992

[3]. MacKay, David J.C. *Information Theory, Inference and Learning Algorithms*, - http://wol.ra.phy.cam.ac.uk/mackay, 1999