Selective Dynamic Compression

S.T. Klein, D. Shapira, E.Opalinsky

Bar Ilan University, Ariel University

August 27, 2019

Overview

Background: Compression System Components

Three Major Components

- model
- encoding process
- inverse decoding process

Compression Techniques

- static . . . model determined during preprocessing and remains unchanged
- adaptive

Dynamic Compression - One-Pass Scheme

Encoder and decoder maintain same model which responds to the local changes and the model **constantly** gets updated.

Adaptive compression usually consist of three main steps for **each** processed symbol:

1. **read** *the following symbol*;
2. **encode** *according to the current model*;
3. **update** the model *(increment the frequency of the currently read symbol)*.

How frequently the model should be updated?

- What is motivation for update with every character read

How frequently the model should be updated?

- What is motivation for update with every character read

- **more accurate** the probabilities

  $\Downarrow$

- **better approximation** for the "future"

How frequently the model should be updated?

- What is motivation for update with every character read

- **more accurate** the probabilities

  $\Downarrow$
- **better approximation** for the "future"

- *Statistic observation, which is* **not necessarily true**
- We propose: update the model only **selectively**

Selective Dynamic Encoding

Algorithm 1: SELECTIVE-ENCODE

SELECTIVE-ENCODE($T = x_1 \cdots x_n$)

```
initialize the model
initialize a random bit generator
for i ← 1 to n do
    encode x_i according to the current model
    bit ← random()
    if bit = 1 then
        Update the model
```

Selective Dynamic Decoding

Algorithm 2: SELECTIVE-DECODE

SELECTIVE-DECODE$(\mathcal{E}(\mathcal{T}))$

```
initialize the model
initialize a random bit generator identical to SELECTIVE-ENCODE
for i ← 1 to n do
    decode x_i according to the current model
    bit ← random()
    if bit = 1 then
        Update the model
```

Specific Variants

We examined the traditional and selective methods comparing:

- *Compression efficiency*
- *Processing time savings (coding/decoding)*

We considered the 50MB file *English* from Pizza&Chili Corpus, which is the concatenation of English text files.
All experiments were conducted on a machine running 64 bit Windows 10 with an Intel Core i5-8250 @ 1.60GHz processor, 6144K L3 cache, and 8GB of main memory.

Arithmetic coding

- One of the most effective compression schemes
- Compression efficiency **approaches the underlying texts entropy**.

- initialized with the interval [low,high) =[0,1),
- which is narrowed for each processed character (according to the characters probability).

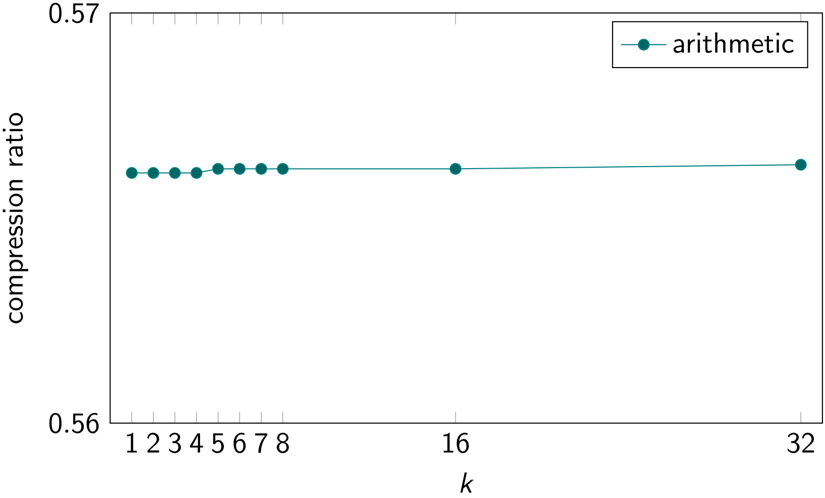
Selective Arithmetic Coding

Practically no loss in compression efficiency.

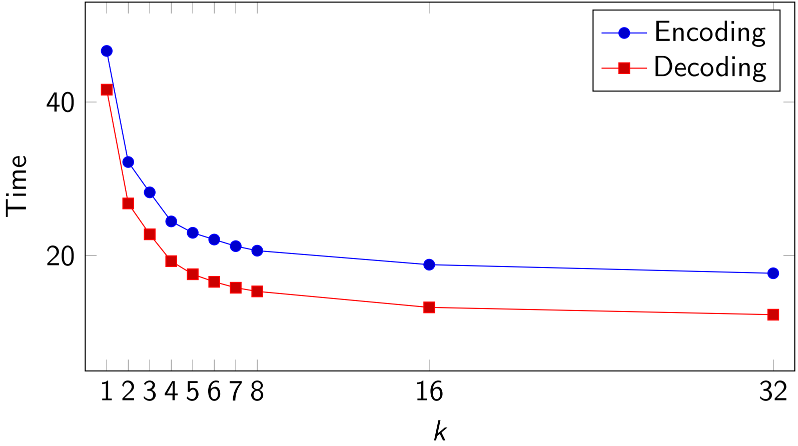
- $P = (p_1, \ldots, p_\sigma)$ - probability distribution of all the characters
- P' - distribution of the characters corresponding to the 1-bits chosen by the random number generator.

- $H(P) \simeq H(P')$ (entropy is almost the same)
- Encoded text size is $n \cdot H(P)$ and $n \cdot H(P')$

Compression Efficiency for Arithmetic Coding



Selective Arithmetic Coding: Processing times



Dynamic Huffman Coding

- The way the model gets updated.

Huf-subset

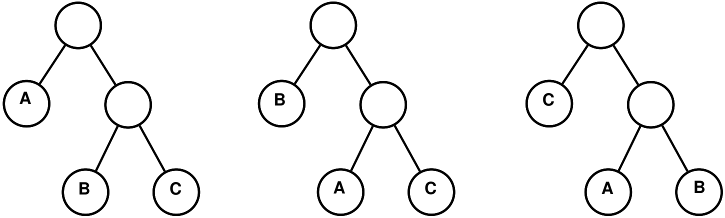
- update the dynamic Huffman tree by advancing the frequency of the **current character only**
- using Vitter's Algorithm

Huf-full

- update according to the changes in the frequencies of *all* **the characters seen since the last update**.
- generate a static Huffman tree from scratch

Selective Dynamic Huffman (Vitter) compression quality

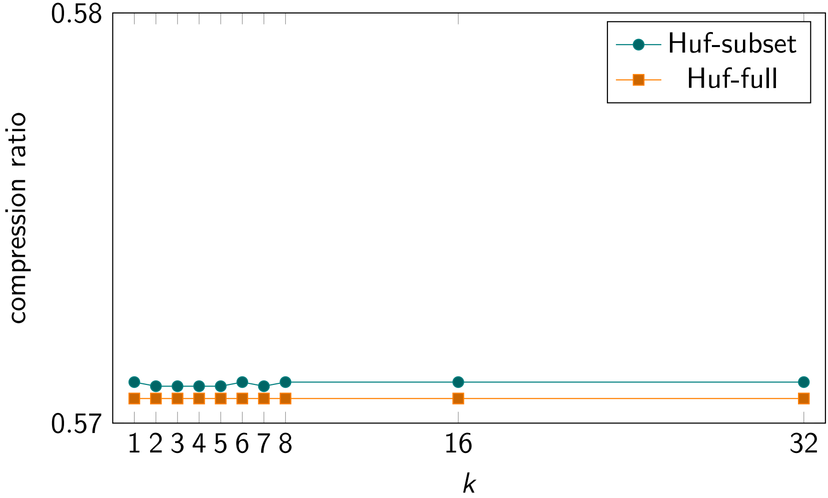
$T = \mathtt{B}\{\mathtt{CCBB}\}^t$ for some positive integer t.



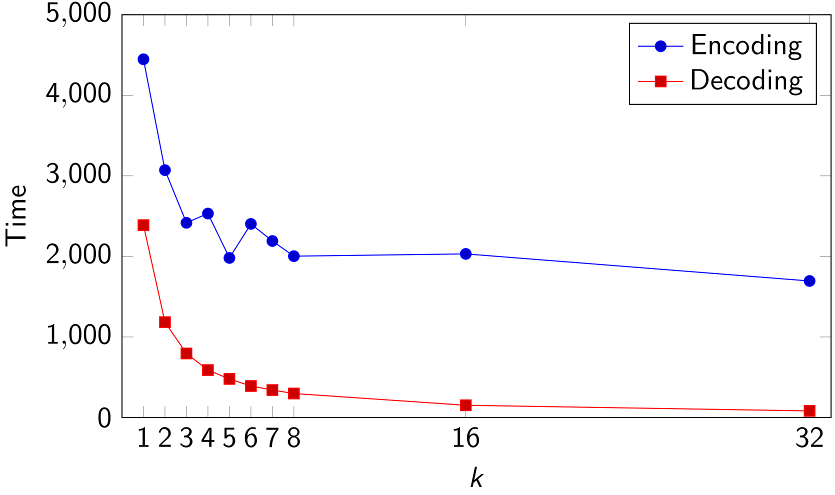
(a) (b) (c)

Example for which selective Huffman coding produces a file $\frac{3}{4}$ of the size of that constructed by standard Huffman.

Compression Efficiency for Huffman



Selective Dynamic Huffman: Processing times



LZW

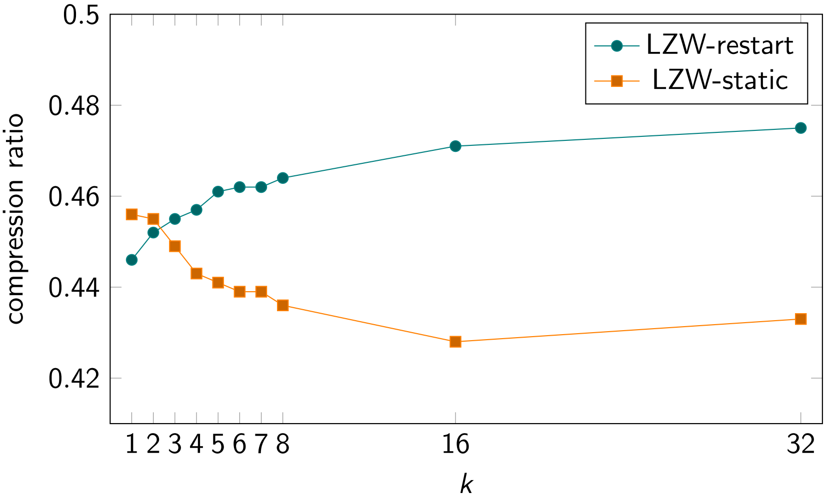
- LZW [T.A. Welch, 1984] is a *dictionary* method
- Dictionary properties:
  - Initialized by the single characters of the alphabet.
  - Updated dynamically by adding newly encountered substrings.
  - Starts with the size of 512 (2^9) - alphabet of ASCII symbols,each encoded by 9 bits.
  - Once filled up its size it is doubled to 1024(2^{10}) entries, each encoded by 10 bits.

LZW

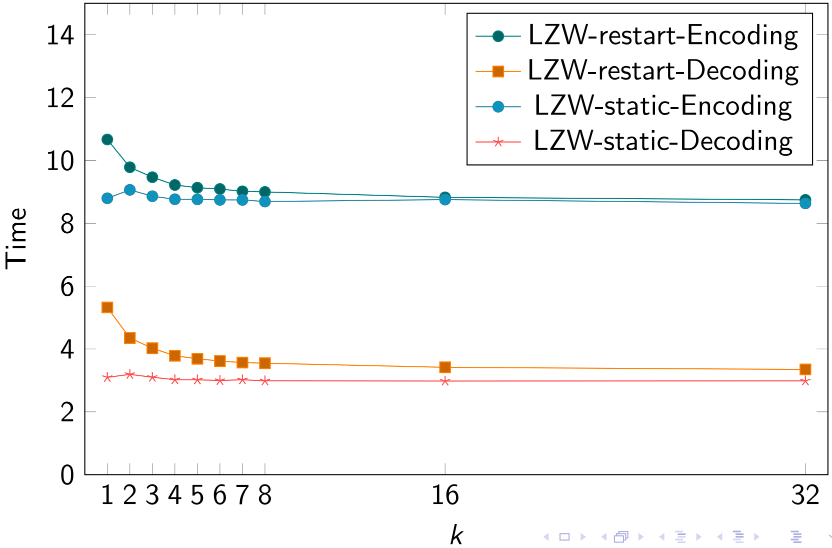
When the dictionary grows up to predetermined maximal size (2^{16} in our implementation), we consider two variants:

- restarting the dictionary from scratch (**LZW-restart**)
- considering the dictionary as static (**LZW-static**).

LZW: compression ratio



LZW: Processing times



Selective Dynamic Coding Cryptographic Application

In [S. T. Klein and D. Shapira, 2017], for arithmetic coding

- a secret key K shared only by encoder and decoder.

Using different keys yields completely different output files, and there seems to be no easy way to decipher the message without guessing K, yet the sizes of the compressed files were practically unchanged for different keys, (1-bit density was kept at $\frac{1}{2}$).

Results for Random vs Traditional

	Compression ratio		Encoding time		Decoding time	
	Trad	Rand	Trad	Rand	Trad	Rand
Huf-subset	0.571	0.571	4447	2789	2388	1190
LZW-restart	0.452	0.446	10.667	9.781	5.321	4.348
LZW-static	0.456	0.454	8.796	8.771	3.096	3.097
arithmetic	0.5661	0.5661	41.61	27.11	46.65	32.39

References

Jeffrey Scott Vitter (1987)

Design and analysis of dynamic Huffman codes, *J. ACM*, 34(4), 825–845

S. T. Klein and D. Shapira (2017)

Integrated Encryption in Dynamic Arithmetic Compression, *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, Proceedings*, 143–154

T.A. Welch (1984)

A Technique for High-Performance Data Compression, *IEEE Computer 17(6), 8–19*

I.H. Witten, R.M. Neal and J.G. Cleary (1987)

Arithmetic Coding for Data Compression, *Commun. ACM, 30(6),520–540*