Writing a competitive BZip2 encoder in Ada, from scratch, in a few days

https://alire.ada.dev/crates/zipada

Gautier de Montmollin

Swiss Ada Event 2025, Schaffhausen

Zip-Ada
bzip2

Web links

Zip-Ada

Zip-Ada free, open-source, full Ada data compression library

Alire Crate: https://alire.ada.dev/crates/zipada

Home page: https://unzip-ada.sourceforge.io/

Sources, site #1: https://github.com/zertovitch/zip-ada

Sources, site #2: https://sourceforge.net/projects/unzip-ada/

Motivations for a BZip2 encoder:

- fun / challenge / warm-up for Advent of Code 2024
- weather
- someone did a documentation in 2016 (20 years after the software!)
- fill a gap in the Zip-Ada compatibility grid:

			Zip-Ada	
Format	**Format #**		**Compress**	**Decompress**
Store	**0**		**v.22**	**v.1**
Shrink	1		v.22	v.1
Reduce 1 .. 4	2 .. 5		v.29	v.1
Implode	6		never	v.1
Deflate	**8**		**v.50 (v.40-49: limited)**	**v.1**
Enhanced Deflate	9		never	v.30
BZip2	12		v.60	v.36
LZMA	14		v.51	v.47
PPMd	98			
Zstandard	93			

Zip-Ada

bzip2

Expectations (low):

- BZip2 compresses few kinds of files better than, for instance, LZMA
- BZip2 compression scheme is mostly “mechanical”: on most steps, there is only one single possible encoding
- BZip2 is a weakened version of BZip1 (old patent issues).

Results: two very good surprises!

Zip-Ada

bzip2

BZip2 is very simple.

1. Input: a "large" block of data (<= 900 KB)

2. The entire block is processed "**off-line**"
   - Run Length Encoding (2x)
   - Burrows-Wheeler Transform (**b**lock-sorting)
   - Move To Front
   - Entropy coding (Huffman)

3. Output of the compressed block.

Zip-Ada

bzip2

BZip2 is very simple.

→ simple source code as well (separate steps):

```
procedure Encode_Block (dyn_block_capacity : Natural_32) is
  ...
begin
  --  Data acquisition and transformation (no output):
  RLE_1;
  BWT;
  MTF_and_RLE_2;
  Entropy_Calculations;

  --  Now we output the block's compressed data:
  Put_Block_Header;
  Put_Block_Trees_Descriptors;
  Entropy_Output;
end Encode_Block;
```

Zip-Ada

bzip2

Run Length Encoding #1

a	⟶	a
aa	⟶	aa
aaa	⟶	aaa
aaaa	⟶	aaaa[0]
aaaaa	⟶	aaaa[1]
aaaaaa	⟶	aaaa[2]
…		

1	⟶	1
2	⟶	2
3	⟶	3
4	⟶	5
5	⟶	5
6	⟶	5
…	⟶	5
259	⟶	5

Zip-Ada bzip2

Burrows-Wheeler Transform

```
Mary had a little lamb, its fleece was white as snow
ary had a little lamb, its fleece was white as snowM
ry had a little lamb, its fleece was white as snowMa
y had a little lamb, its fleece was white as snowMar
 had a little lamb, its fleece was white as snowMary
had a little lamb, its fleece was white as snowMary 
ad a little lamb, its fleece was white as snowMary h
d a little lamb, its fleece was white as snowMary ha
 a little lamb, its fleece was white as snowMary had

…
```

```
a little lamb, its fleece was white as snowMary had
as snowMary had a little lamb, its fleece was white
fleece was white as snowMary had a little lamb, its
had a little lamb, its fleece was white as snowMary
its fleece was white as snowMary had a little lamb,
lamb, its fleece was white as snowMary had a little
little lamb, its fleece was white as snowMary had a
snowMary had a little lamb, its fleece was white as
was white as snowMary had a little lamb, its fleece

…
```

Sorting

Reversible!

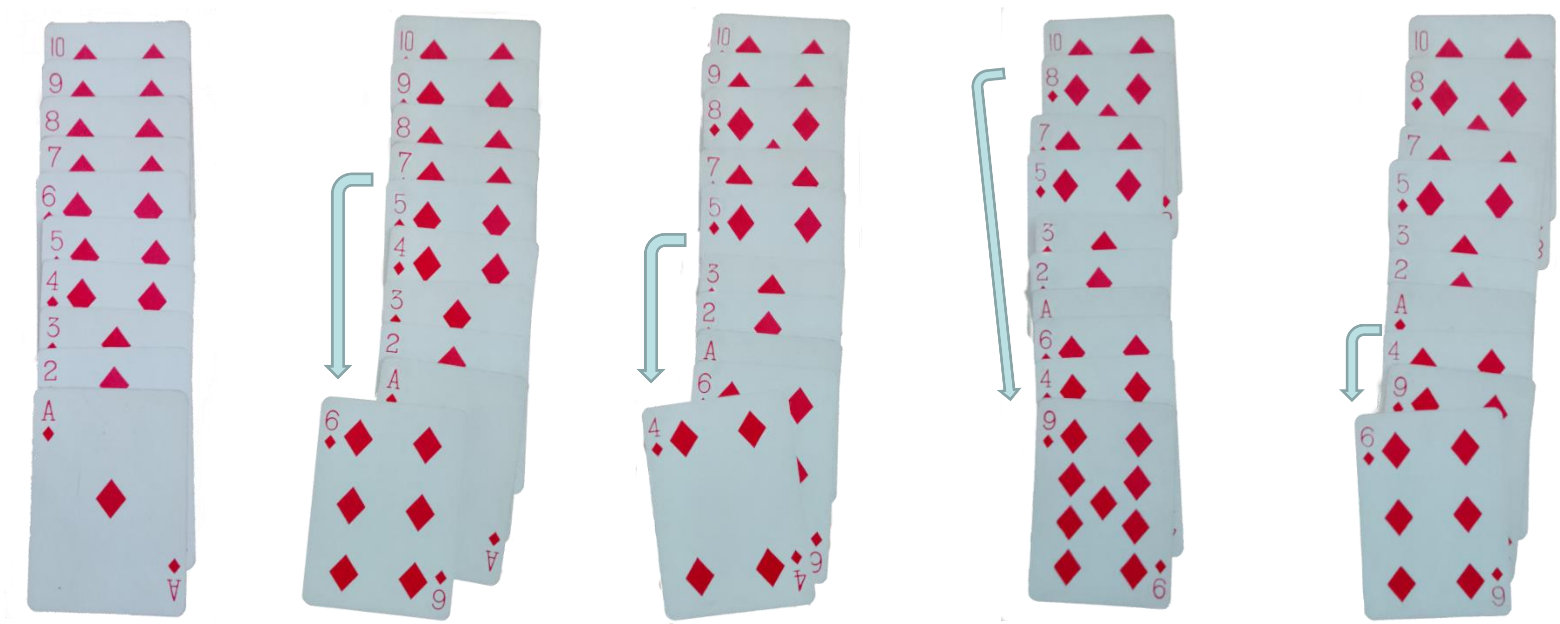
Burrows-Wheeler Transform – continued

Output of the Ada source itself, bzip2-encoding.adb (excerpt):

```
2)_____FPUAAEOOA(     E   _______'_'''''_'''''''''__________________________   ___( 8)  ____  __
________RSGAESREIIOUI I   EE ( 6)  ( 4)   ( ______       ( 8)( 10)  ___   ( 6)__ ______    "  _______-
_______( 8)( 8)( 4)( 10) ( 8)        ( 4)( 4)( 8)( 8)( 8)______  _   _   ..
HOIUURNUWWCWNWWWWNIMMMMMMMMMMMMMI      OOEOUIRNCCI  NFFFNIII___________   ...         ___( 6)( 10)(
12)( 8)( 8)( 8)( 8)( 4)( 2)( 4)( 6)( 6) ( 8)     ___     ____________LCRRRRRRP BOOA.. ...( 4)( 0)( 4)
(((( ( (   (._______ _________EO''_ __   __ BTTT   R  SBBBBBBB      -( 2)  ( 0)  ( 0)ENNTTTPT___ -(
0)BBBBBBBBBBBBB-
'11EkkkeeeeeeeeeEEE22E0EEEEdrrddldrrlldd lllrdllllllllleeeeeleellllreledlllrlddlrellled0kkkdkkkkNNNeryer
erryxxxxxxtFeettthhteFgtgthFtghtFFdddrrNNNhhhhnnnnnnttttttttttttttttttttttttttttttttteeeettttttttttedee
eyyywwwwllllllllllllyyyyyyyyyyyyyyyyyyyyyeddccslsslsssldddeeeesss22eeeeeetmmmdddddttttooeeeeeeeeekmmk
mkttteeetttrrttennnttttttthhtw22ggeeetttyyyygggggggnyyyttdddFFrrrrtrtttrrrrtrtttrrrrtrrrrryyyyyyyygggdddte
eeeeeeeeettteeeeeemmgggyyyyytgtgyrrrrrrreeeeeeodgttttgtgttteeekkketxyyxsettsennnnnnnnnnnnnnnhhheedddnn
nxFFFeeenttttwwttttttwwttttwwwwtttxnnnnnlllllbtttttgggggggrrrnnkkkkkkkknhnhtttqqqnttnrqtqtntrrnnnnrrttff
ffffffffffffxxxxxxxxxxxxxxxxxxeeyyyyyyyyyyyyyyyyyyyyyyyyytttccrrrrrrrrrrrrrrrrrrrrrrrrrrrrcrrsssssss
sssssssssssssssssssssssskkkkkdkkddkkkddkkktffffffffffftftftttrrrrrrrtttxxmplllplpppleekmeefllllllfrffl
lllllfrfrlllllllfflffffffffrfrffrllllllfflllllfffffffllllrlrttttttttttttttttttllllllsslssslllssllttttnnffffff
fffffffffffeeeeeeeeeeeeeeeeeettttttttttttttttttttttttttttttttttttttttyyyyyynnrrryyppptttgggppeeeeeppppptt
tttttbbbbbkkkkkmmmmmmttttllttttoototttteekkkktkkkkkpknpppkknkpkkkkpppppppppppppppeeeffffyyffyffttttttttllll
lllcnhhnchceeqqqnnnnnnnnnnnnnnnnnnnnnnnnnnnrrrrrrrrrreeeeeeeeeeeeee
```

Zip-Ada
bzip2

Move To Front

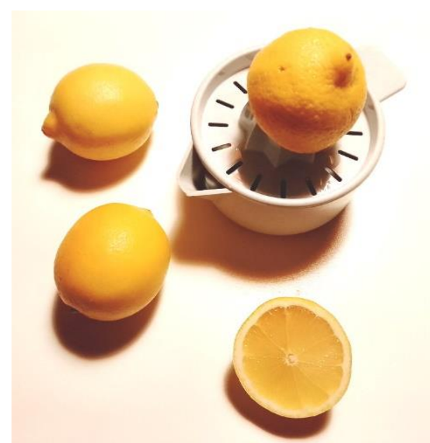


What we have to send →	Card: “6”	Card: “4”	Card: “9”	Card: “6”
What we actually send →	Index: 6	Index: 5	Index: 9	Index: 3

Final step: entropy coding with Huffman trees

Not mechanical. You have up to 6 trees, *freely* defined, that can be *freely* chosen for each string of 50 symbols (the output of Move To Front)

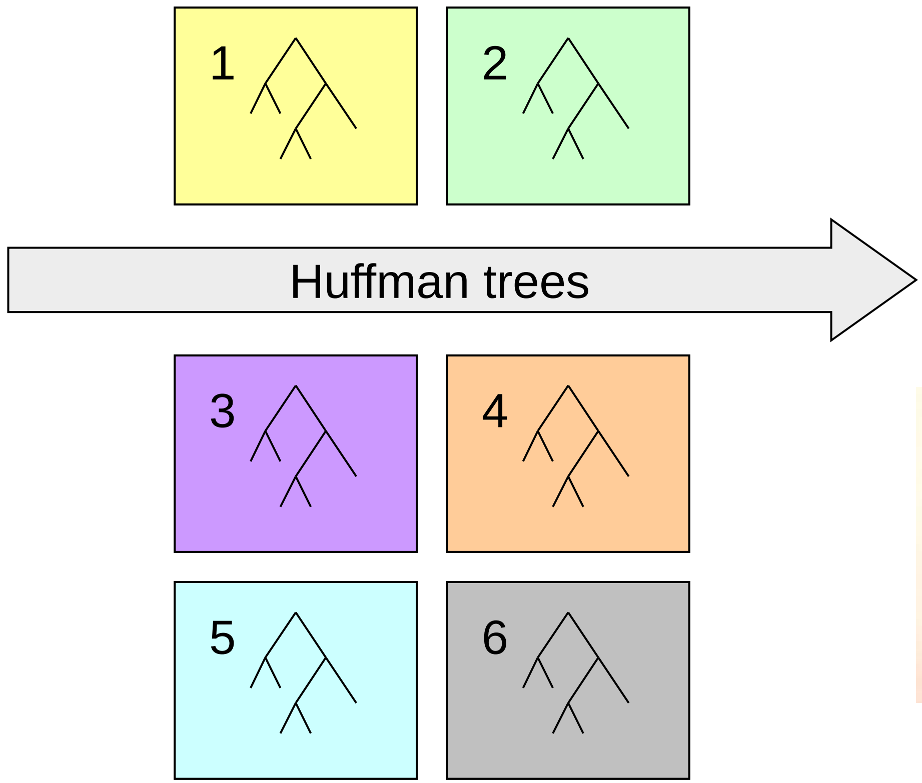
⟶ Room for **optimization**!



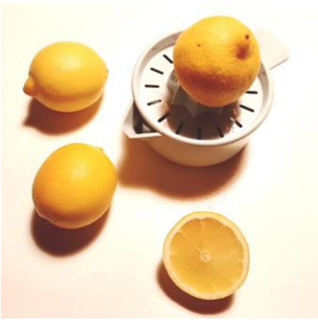
Zip-Ada bzip2

Entropy coding choices

50 symbols
50 symbols
50 symbols
50 symbols
50 symbols
50 symbols
⋮
<= 50 symbols



Binary bzip2 data stream



Each symbol is of an alphabet of max 258 elements

Choice of Huffman trees

50 symbols

50 symbols

50 symbols

50 symbols

50 symbols

50 symbols

▪
▪
▪

<= 50 symbols

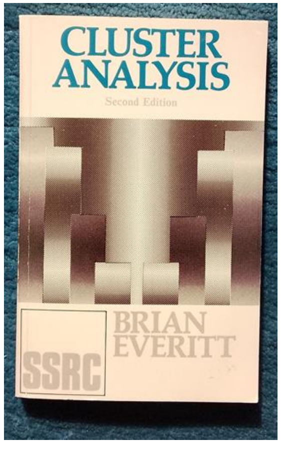
How to allocate the strings of data to the Huffman trees?

For shortest encoding, depends on the Huffman trees.

How to define convenient Huffman trees?

Depends on the data: the strings.

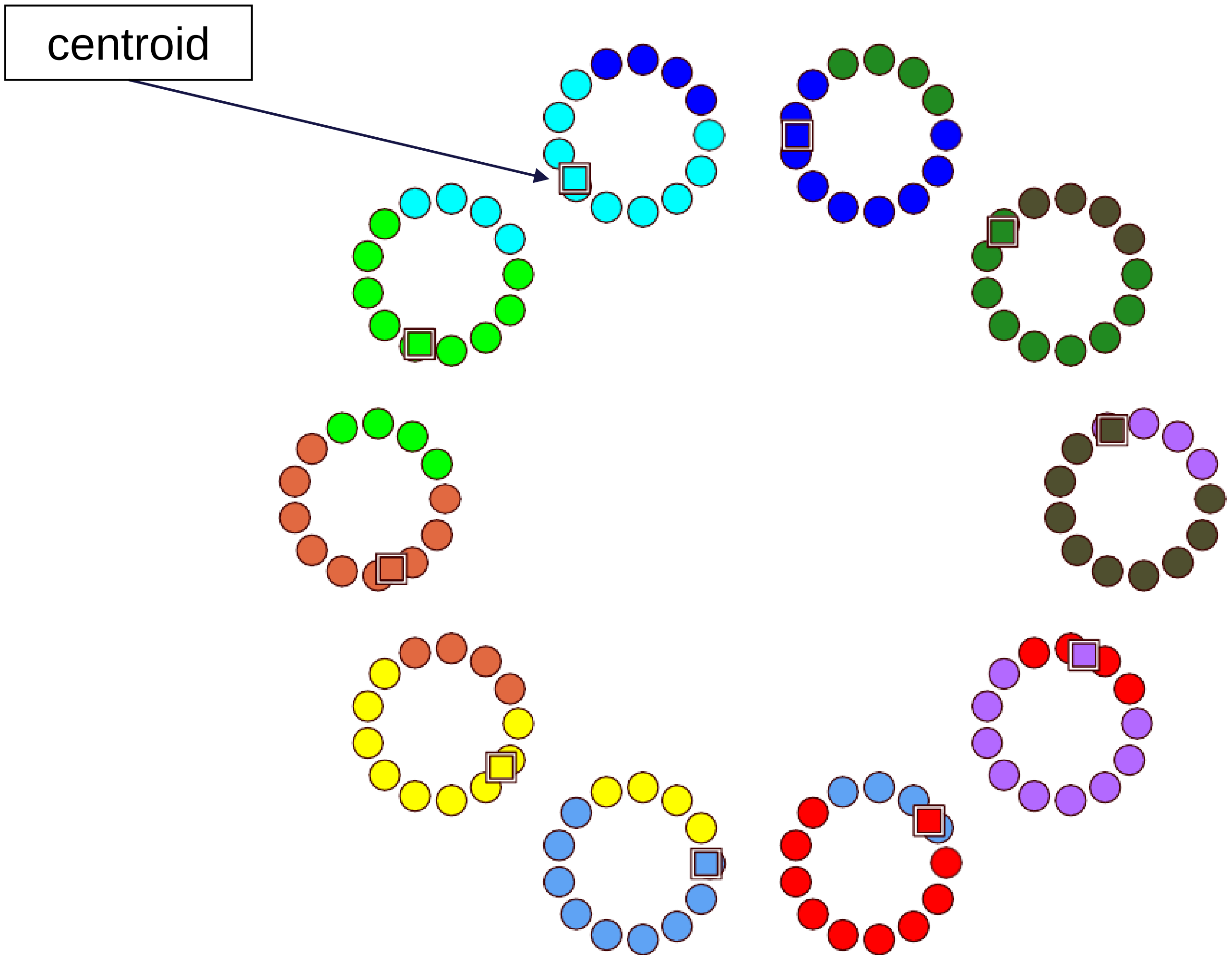
Actually, it's a



problem!

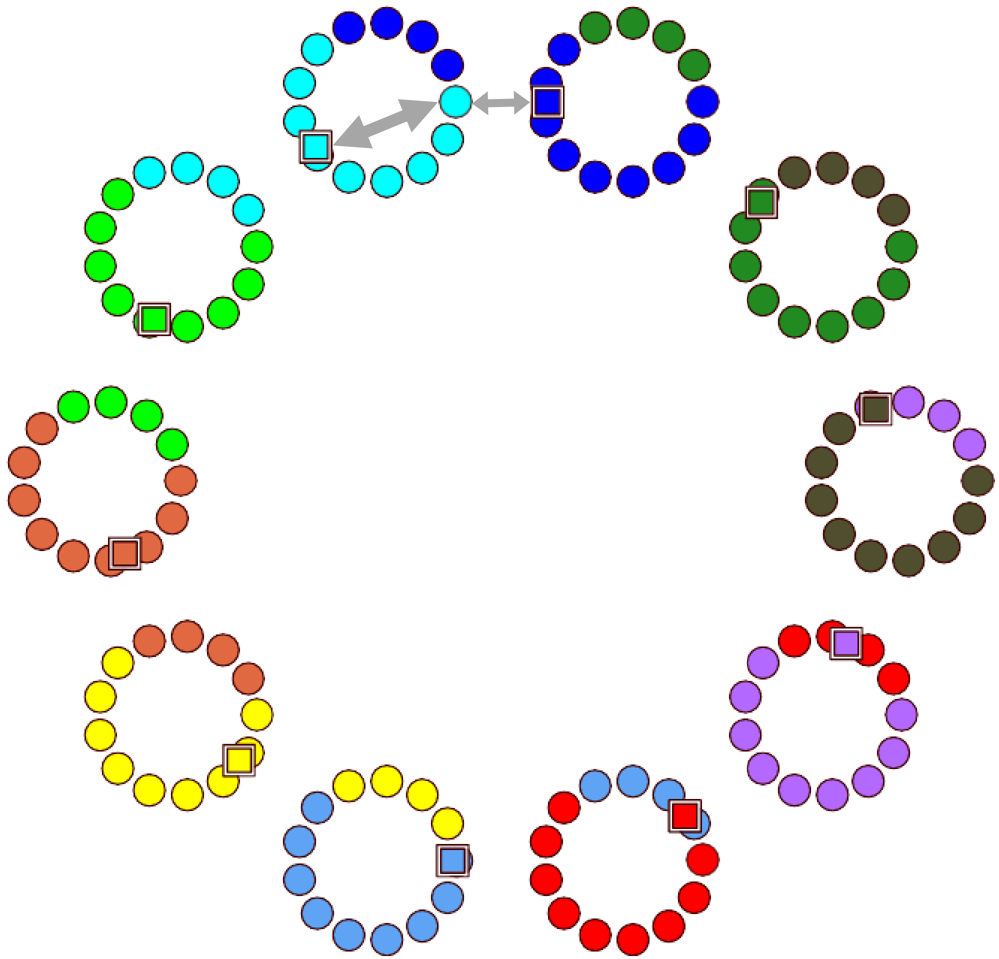
Partitioning Clustering: initial clustering and *n* iterations of optimization

Clustering: 2D example. Initial ("imperfect") clustering.

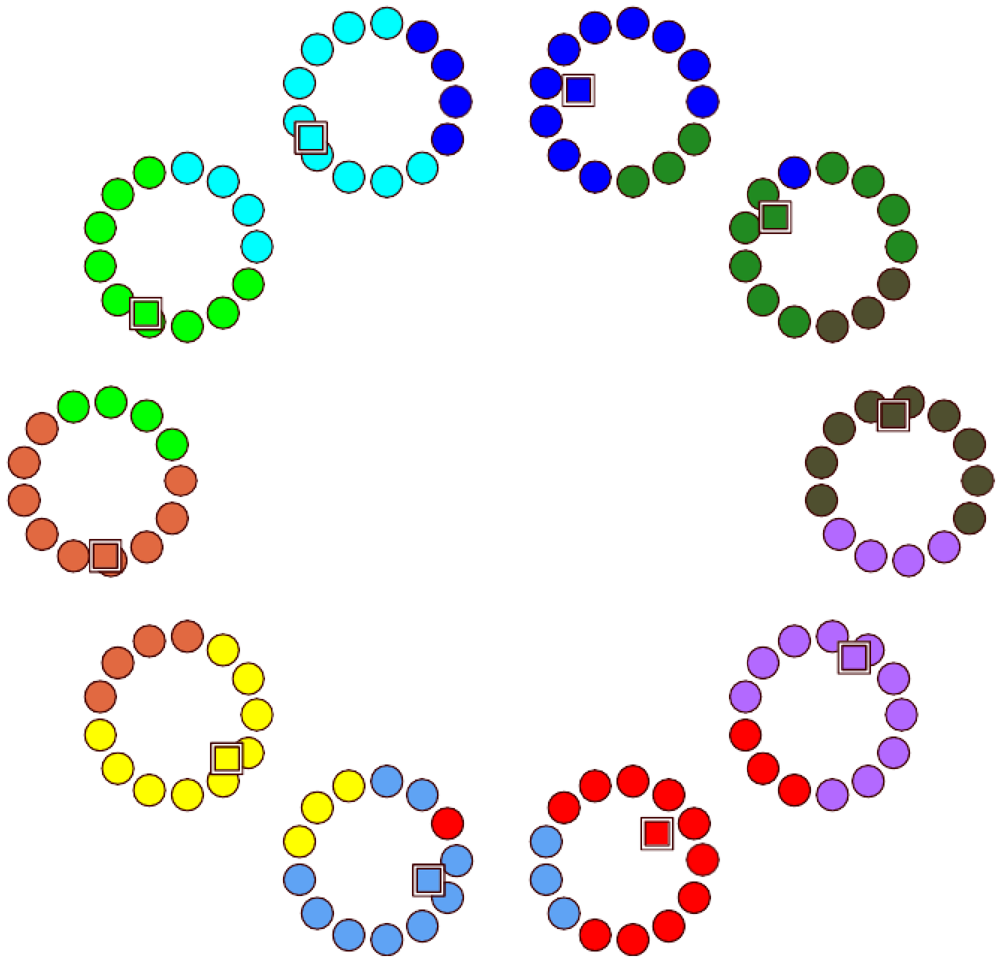


Demo in: [Ada PDF Writer]

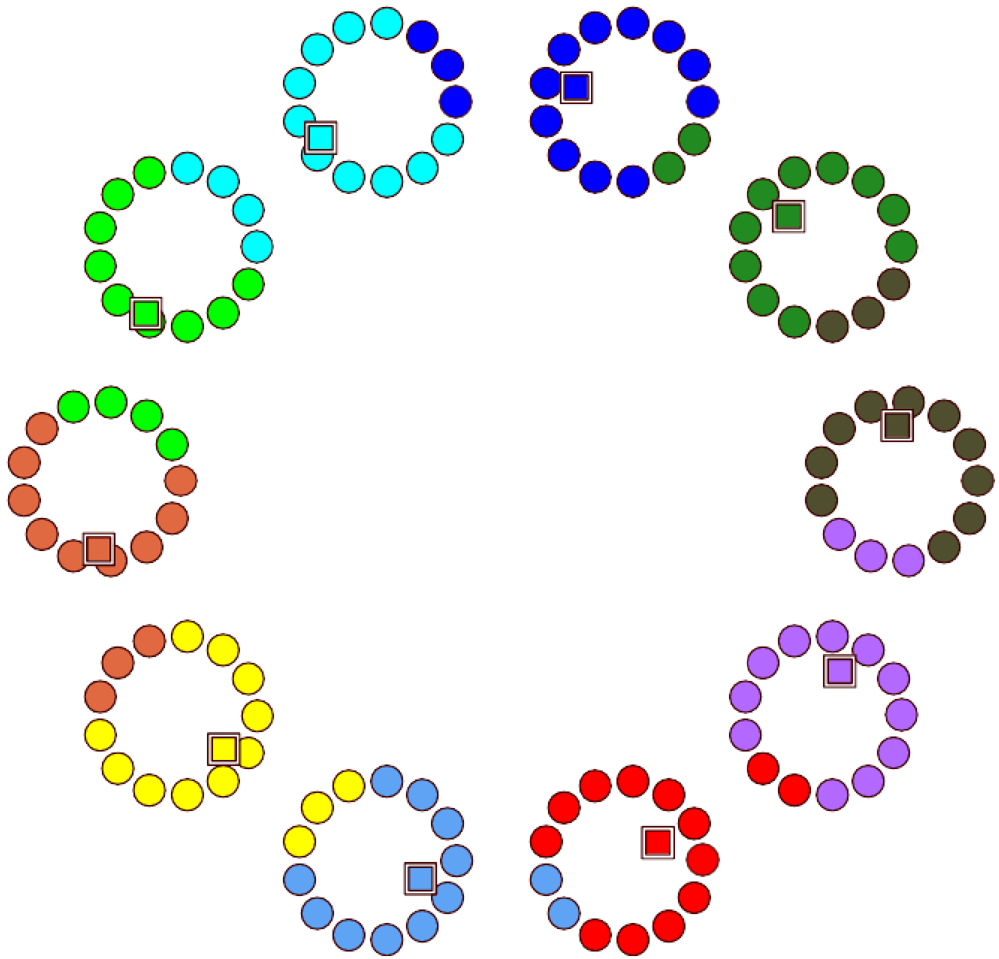
Clustering: 2D example. First reallocation round.



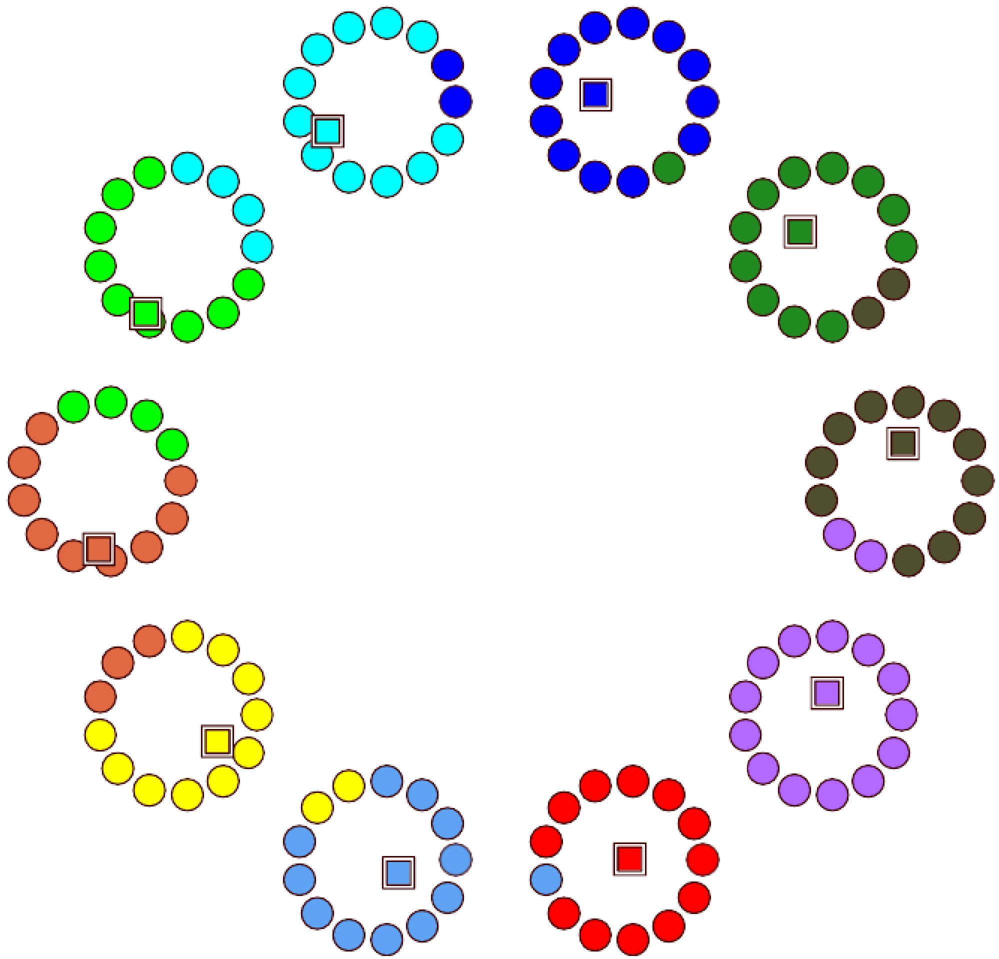
Clustering: 2D example. k-means, iteration 1



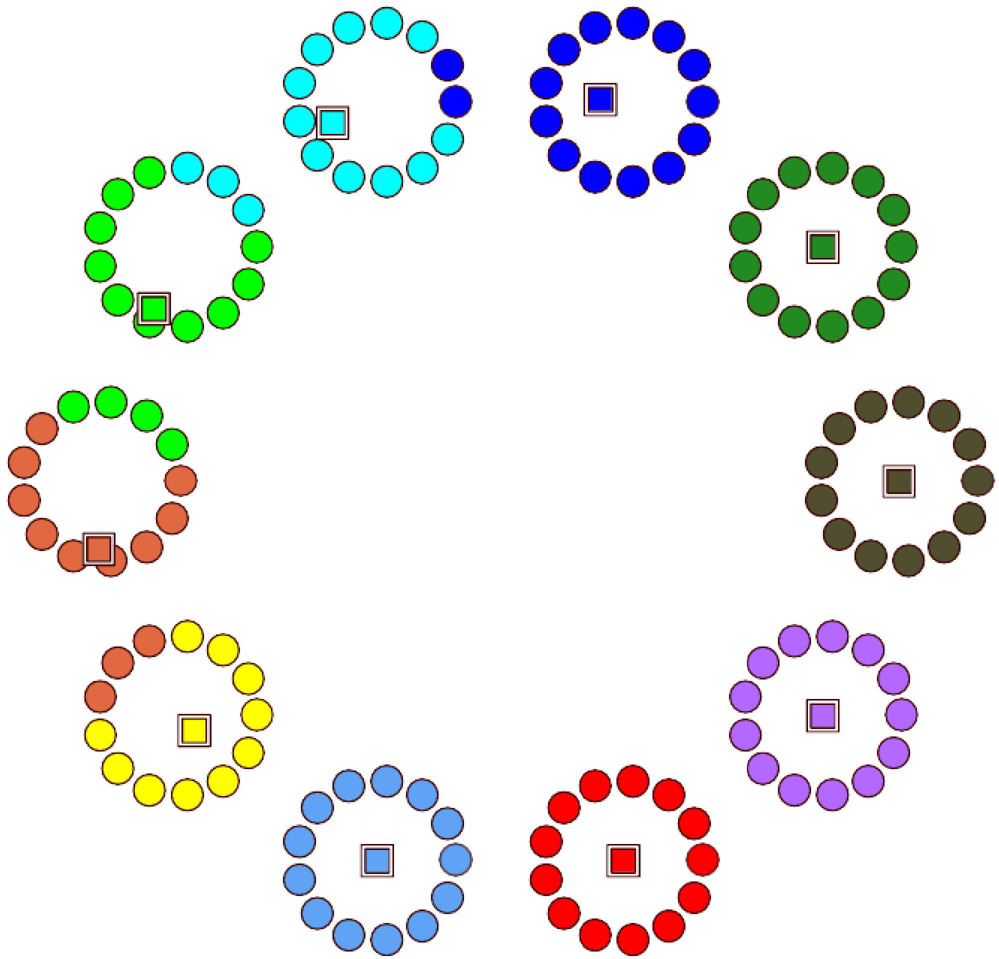
Clustering: 2D example. k-means, iteration 2



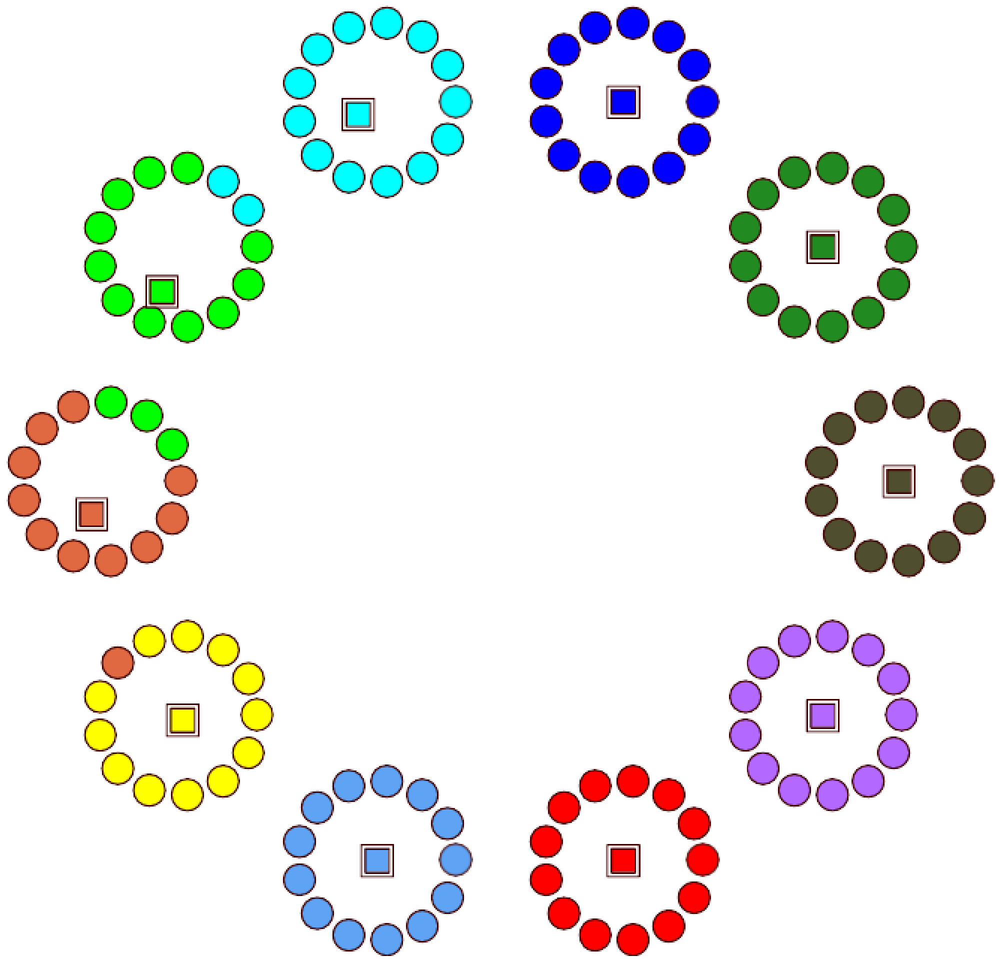
Clustering: 2D example. k-means, iteration 3



Clustering: 2D example. k-means, iteration 4

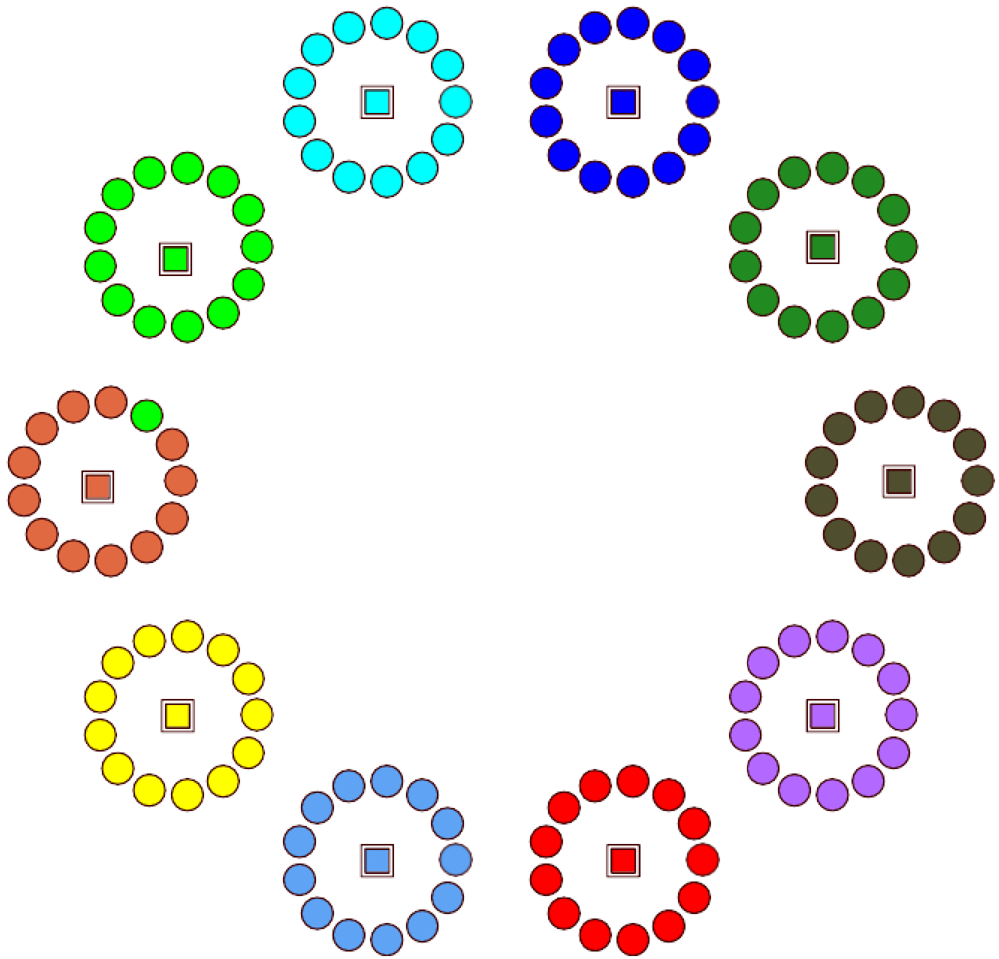


Clustering: 2D example. k-means, iteration 5

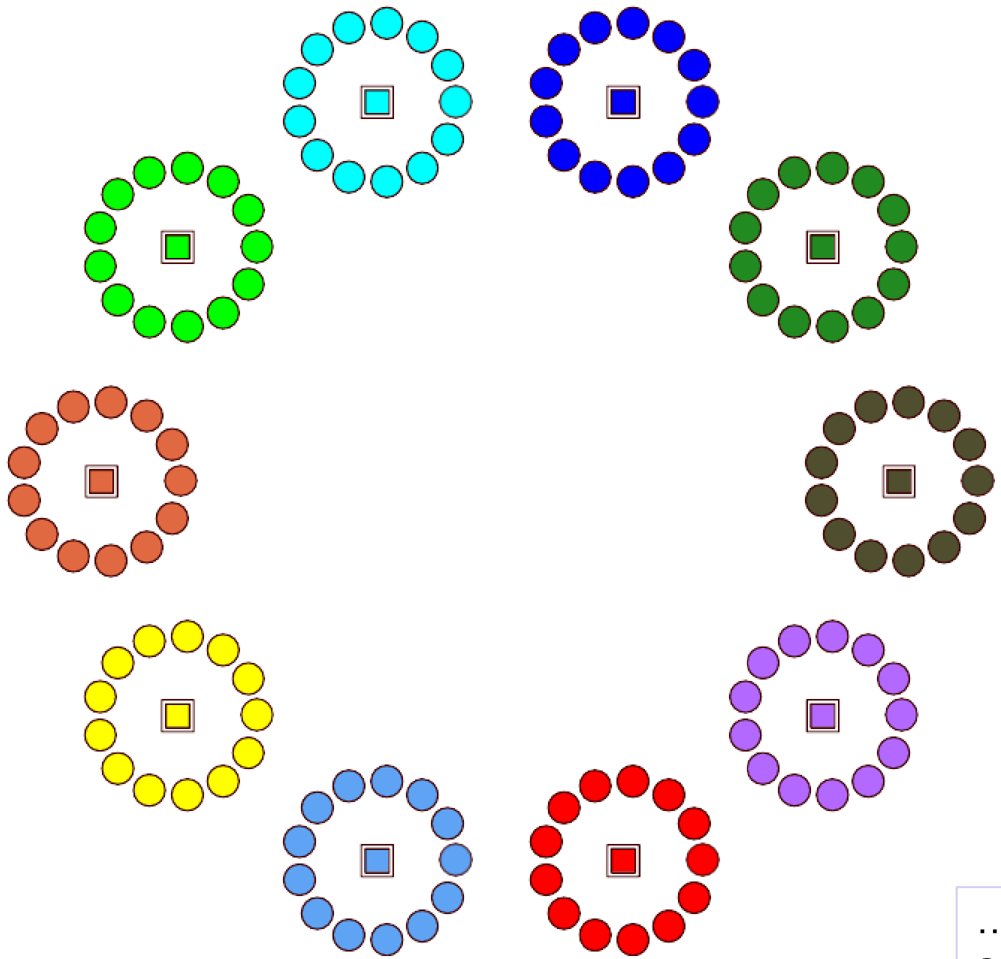


Clustering: 2D example. k-means, iteration 6

Zip-Ada
bzip2

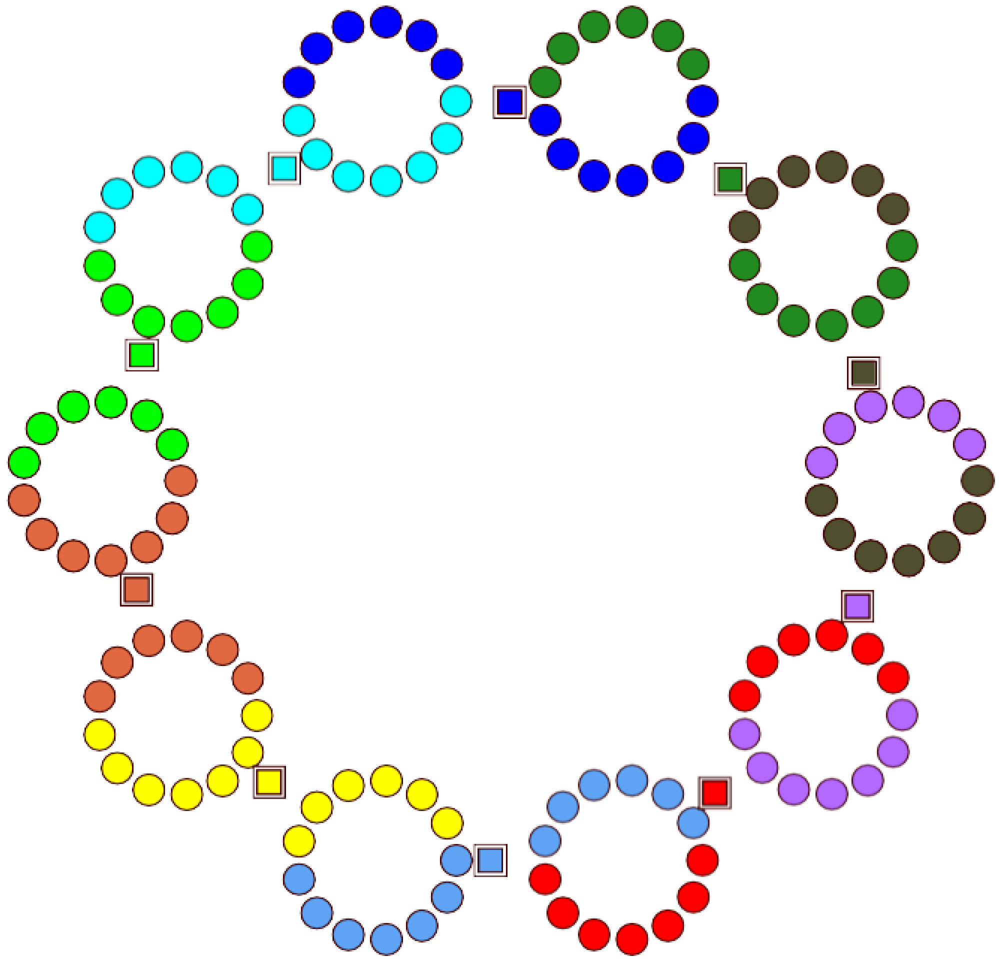


Clustering: 2D example. k-means, iteration 7 (final)

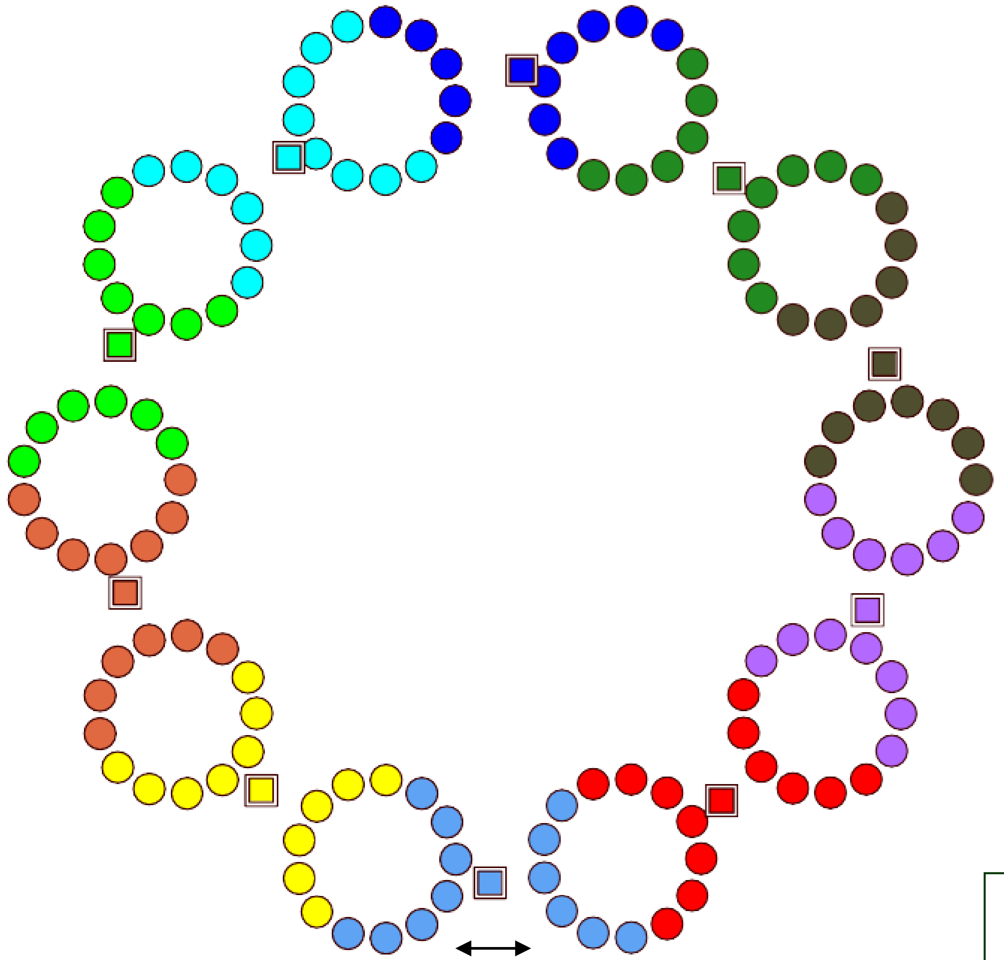


…but this example is a “nice weather” case

The choice of the initial clustering is crucial.
If we start with:



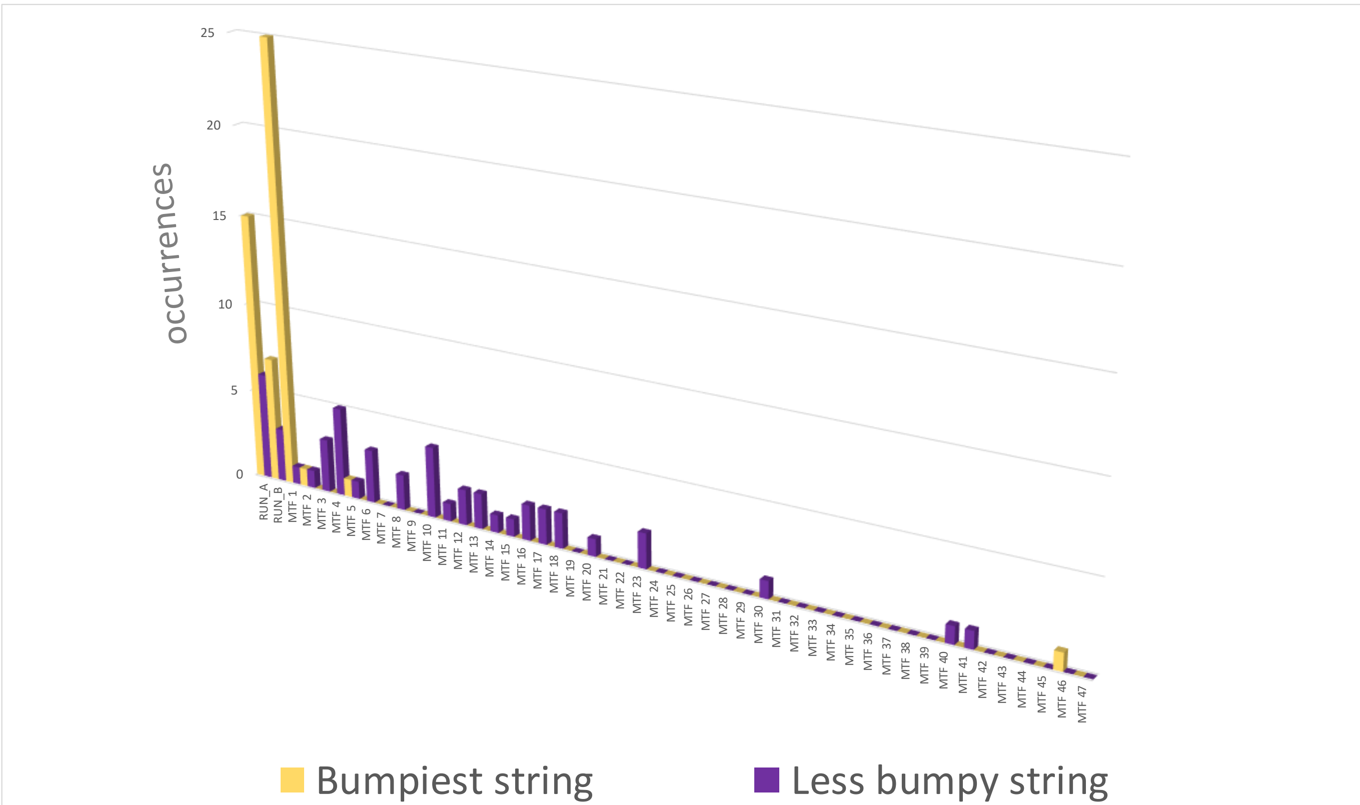
… we are stuck after two steps with this:



→ **Initial clustering** is key.

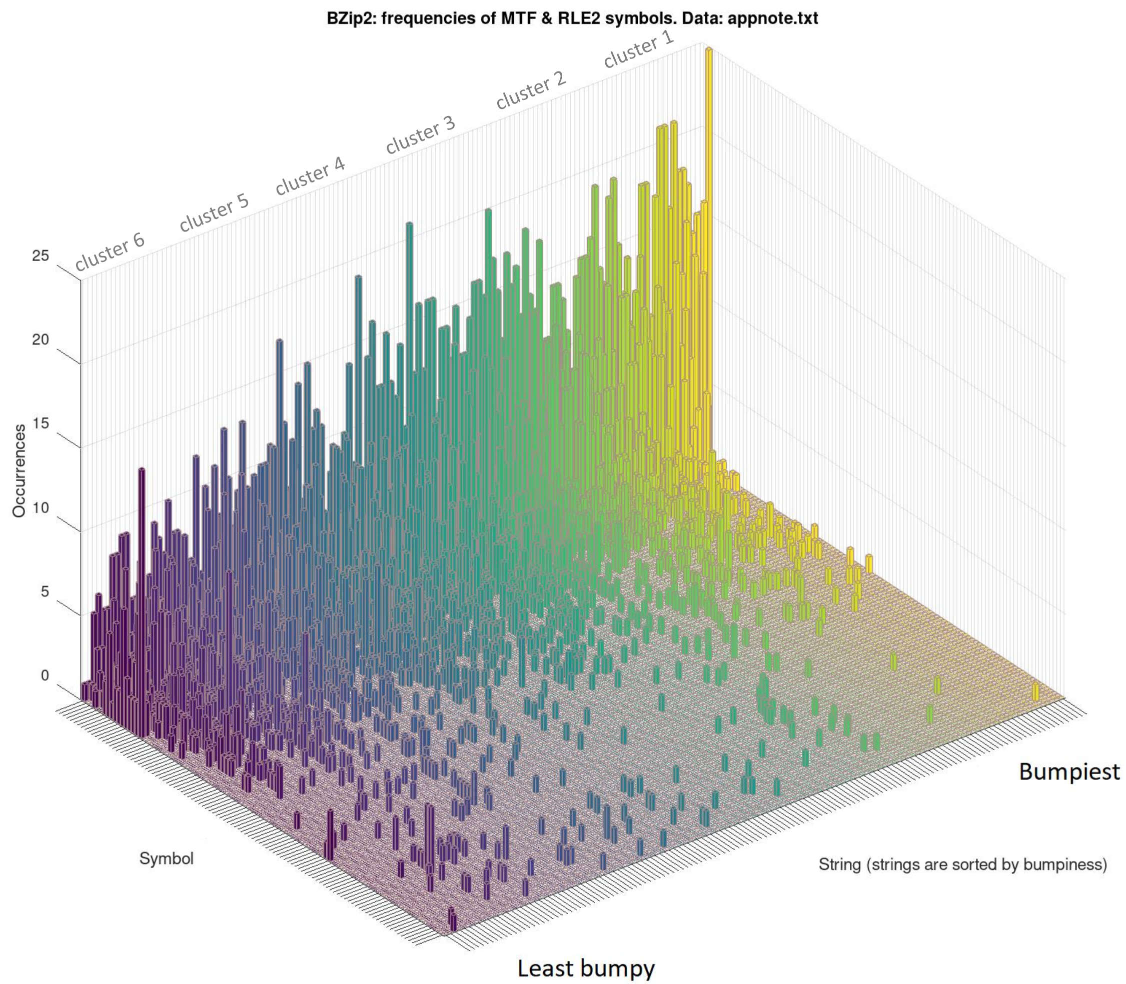
Especially, for BZip2, we have more than 2 dimensions: up to 258 …

Initial clustering for BZip2: sorting MTF & RLE2 symbols frequency histograms by “bumpiness”

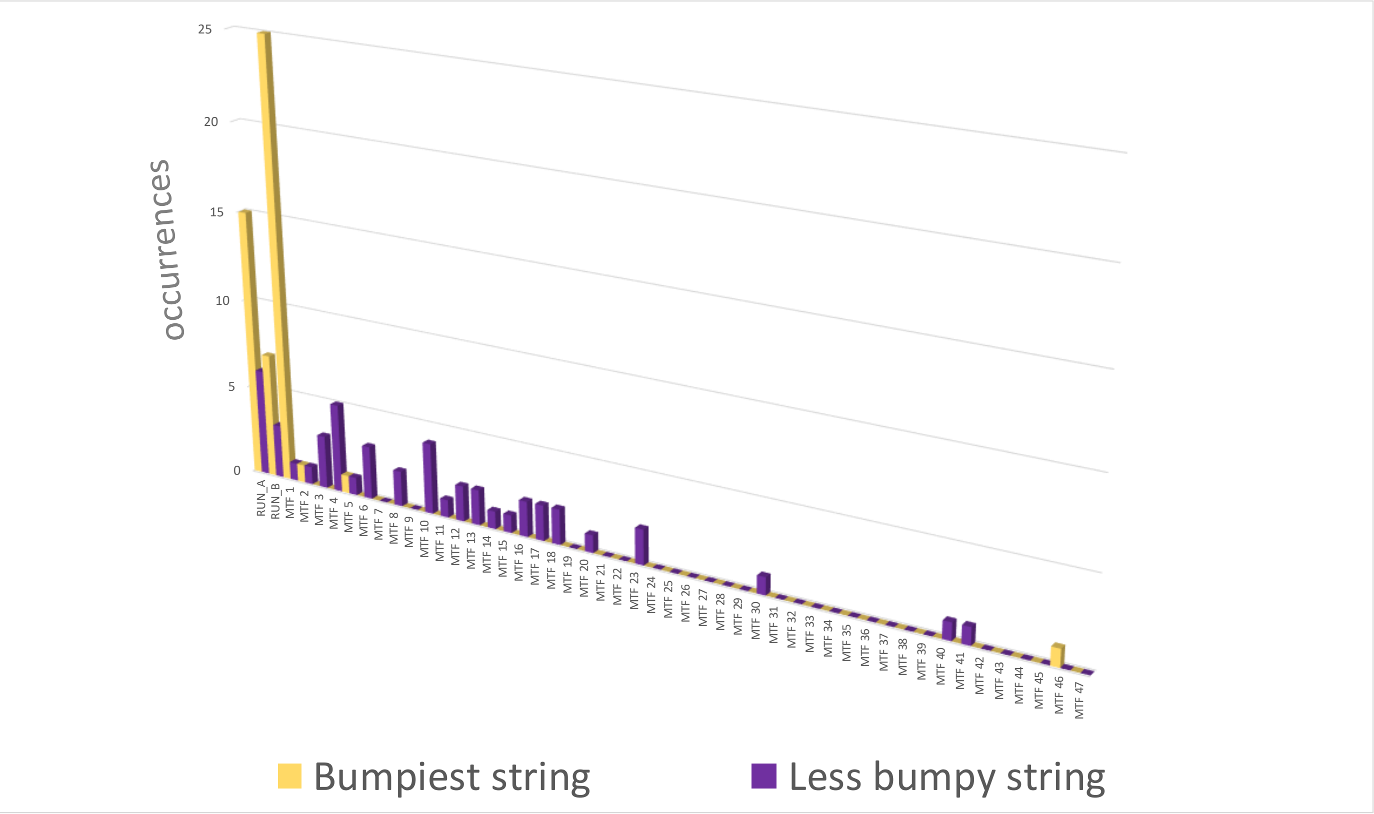


Strings with various “bumpiness” (bumpier: more **redundant MTF** data; input data: **appnote.txt**)

Initial clustering for BZip2: sorting MTF & RLE2 symbols frequency histograms by “bumpiness”



Initial clustering for BZip2: “bumpiness” function



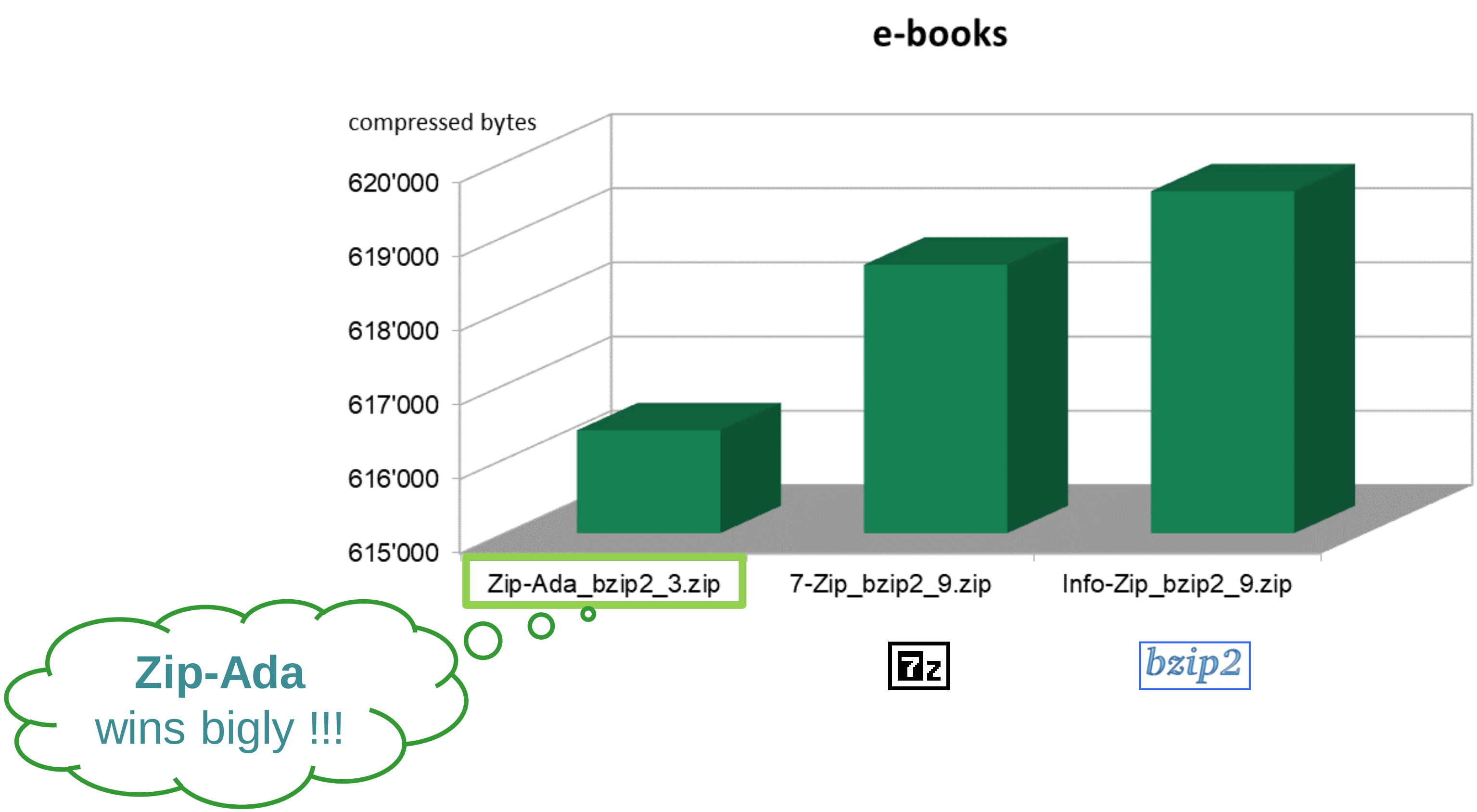
<u>Quiz</u>: how did I define the measure of bumpiness?

Hints:

- all strings (except at most one) have exactly 50 symbols.
- high histogram bars tend to be on the left side because of the Move-to-Front process

Results – first surprise

Zip archive, BZip2 only:

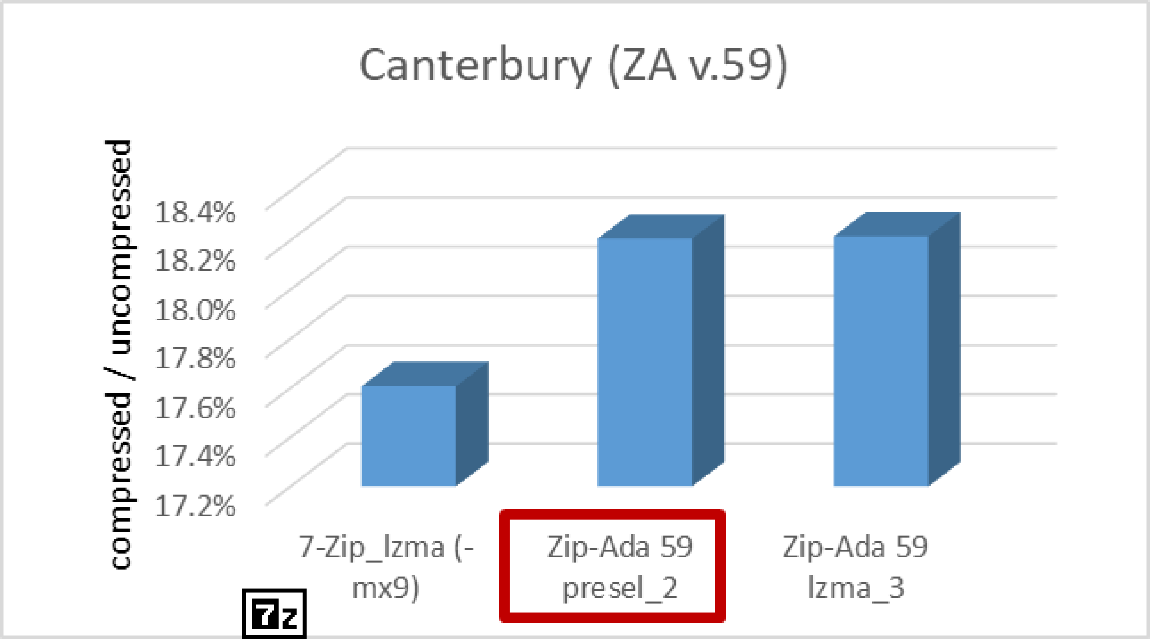


NB: BZip2 is very good with (at least) human-written **texts** and **source code**.

Results – second surprise

Zip archive, multi-format (for Zip-Ada, Preselection_2)

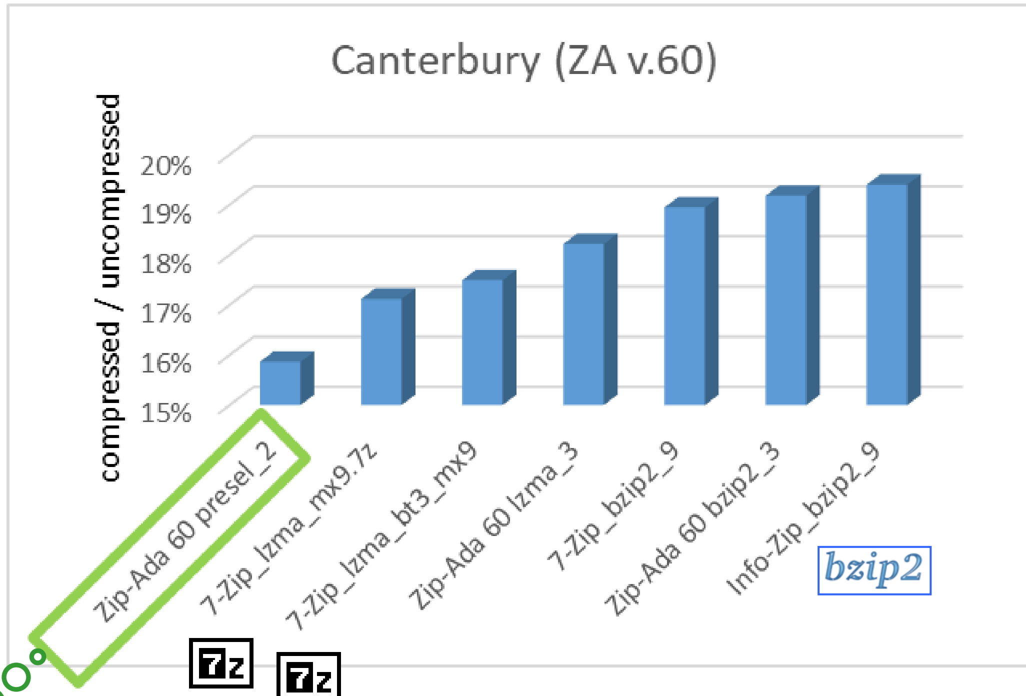
Before:



Results – second surprise

Zip archive, multi-format (for Zip-Ada, Preselection_2), or .7z archive.

After:



Zip-Ada wins here too !!!

Benefits of Ada

Top Ada goodies:

- **Modularity**
- **Very precise typing**
- **Pointer-less programming**. No worries about **accidental deallocation**, **null pointers**, **dangling pointers**, **ownership** (the caller stays always the owner), **lifetimes**, etc.
  In the BZip2 encoder, heap allocation is only used for five large arrays; pointers are used only for allocation & deallocation.

Benefits of Ada – continued

Data compression is very difficult to debug, sometimes impossible.

Ada does its best to help you doing things right the first time and avoiding stupid bugs (you only keep the clever bugs😉). Specifically, it helps you avoiding **omissions**, **duplicates**, **confusions** between types, name **collisions**.

⟶ Use Ada for data compression (and the rest as well)!

Indirect benefit : you can focus on the algorithms!

Benefits of Ada – continued

Zip-Ada

bzip2

Here, some **ranges** picked up from the code (bzip2-encoding.adb):

```
subtype Bit_Pos_Type is Natural range 0 .. 7;
type Buffer is array (Natural_32 range <>) of Byte;
subtype Offset_Range is Integer_32 range 0 .. block_size - 1;
subtype Max_Alphabet is Integer range 0 .. max_alphabet_size - 1;
type MTF_Array is array (Positive_32 range <>) of Max_Alphabet;

type Entropy_Coder_Range is range 1 .. max_entropy_coders;
descr : array (Entropy_Coder_Range) of
          Huffman.Encoding.Descriptor (Max_Alphabet);
entropy_coder_count : Entropy_Coder_Range
   range 2 .. Entropy_Coder_Range'Last;

subtype Alphabet_in_Use is Integer range 0 .. last_symbol_in_use;
type Huffman_Length_Array is array (Alphabet_in_Use) of Natural;
type Count_Array is array (Alphabet_in_Use) of Natural_32;

subtype Selector_Range is Positive_32 range 1 .. selector_count;
type Cluster_Attribution is array (Positive range <>) of Entropy_Coder_Range;
type Value_Array is array (Positive range <>) of Natural;

in_use_16 : array (Byte range 0 .. 15) of Boolean := (others => False);
```

Data dependent!

More @ https://gautiersblog.blogspot.com/2024/11/writing-bzip2-encoder-in-ada-from.html