CS 351 HW4 Write Up

Summary of Files:

bitmap_indexer.py

This is the main file which contains the “create_index “and “compress_index” functions.

Readme.txt

Contains Instructions on running the code.

Summary of how my compression functions work:

def compress_wah_line(dataline, word_size)

This is a function in the bitmap_indexer.py file, which handles compressing a line of bits into wah. There are variables to track the index, runs and run type (“0”s or “1”s). The dataline is looped through word_size characters at a time. If a run is encountered and it is of the same type as the already stored runs or there is not currently a run, then runs is incremented, otherwise the function flushes the currently stored run. If a literal is encountered it is flushed.

There are two helper functions, save_literal(literal), and save_run(run_bit, num_of_runs). These are responsible for flushing / adding onto the compressed output string.

def compress_bbc_line(dataline)

This function compresses a line of bits using BBC. Again we loop through the given dataline 8 characters at a time. This time the runs, and a list of the current literals are tracked. There is a single helper function which is responsible for flushing chunks to the compressed output.

In the while loop flushing happens when: a run is encountered but there are some literals before it, if the number of literals in the list reaches 15, and if the number of runs reaches 32767.

The “flush_compression(padded=False)” function starts by encoding the number of runs in the header byte, it checks if we have a single literal, that has a single “1” and this is not a padded literal (this happens when the last byte of the line gets padded). And in this case this byte gets encoded as a dirty bit, otherwise the number of literals is stored in the header byte.

Analysis of Compressions:

Table of Data

bitmap	sorted_bitmap	compression_method	word_size	total_fill_words	total_literals	compressed_size_bytes
animals	FALSE	BBC	8	59065	140935	1361344
animals	FALSE	WAH	4	318628	214716	1497684
animals	FALSE	WAH	8	76429	152147	1558008
animals	FALSE	WAH	16	14025	92647	1661760
animals	FALSE	WAH	32	1271	50345	1650016
animals	FALSE	WAH	64	26	25382	1626144
animals	FALSE	NONE				1700000
animals_sorted	TRUE	BBC	8	161702	38298	339488
animals_sorted	TRUE	WAH	4	532104	1240	716940
animals_sorted	TRUE	WAH	8	226996	1580	51744
animals_sorted	TRUE	WAH	16	104962	1710	56736
animals_sorted	TRUE	WAH	32	49838	1778	115616
animals_sorted	TRUE	WAH	64	23604	1804	227232
animals_sorted	FALSE	NONE				1700000
animals_small	FALSE	BBC	8	609	1391	13624
animals_small	FALSE	WAH	8	752	1536	15640
animals_small	FALSE	WAH	16	145	927	16736
animals_small	FALSE	WAH	32	13	515	16896
animals_small	FALSE	WAH	64	0	256	16416
animals	FALSE	NONE				17000

For BBC, the number of literals includes the number of dirty bits. This is because there are several cases where a dirty bit is stored like this 00010001, so it is outside of a run. Since there was not a perfect way to count dirty bits, their count is included with the literals.

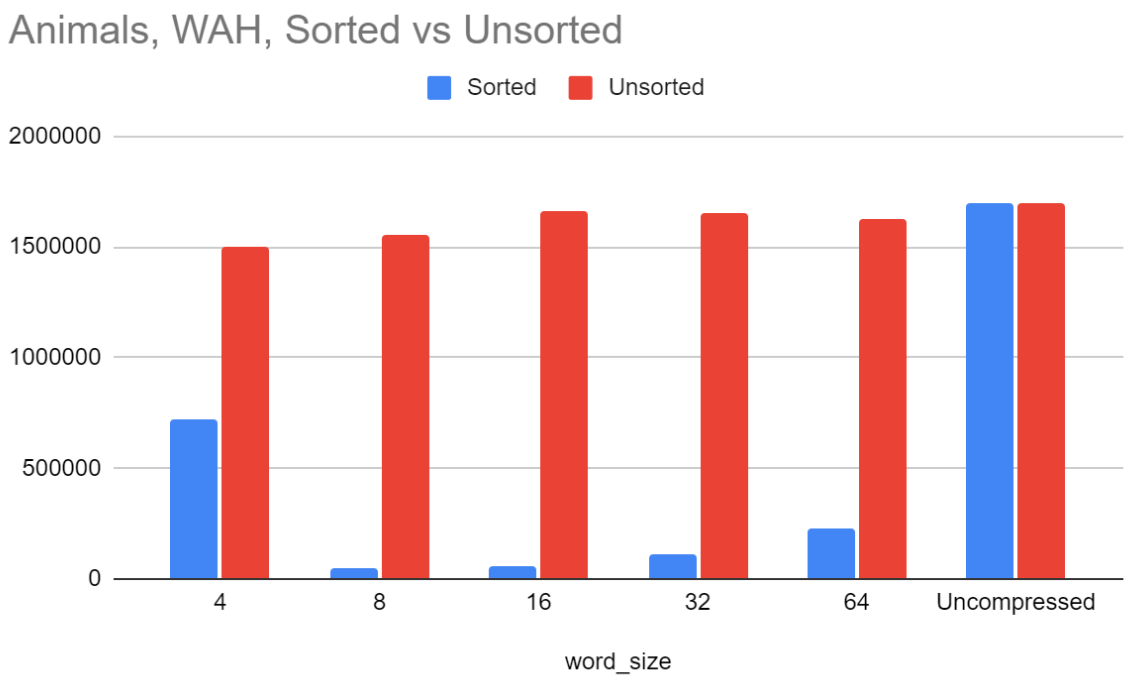
Bitmap indexes vs compressed versions on “animals” bitmap.

The original size of the “animals” bitmap is 1700000 bytes.

In the optimal WAH case where the bitmap is sorted and a word size of 8 is used the compressed size was 51744 bytes. This is 32 times decrease in size.

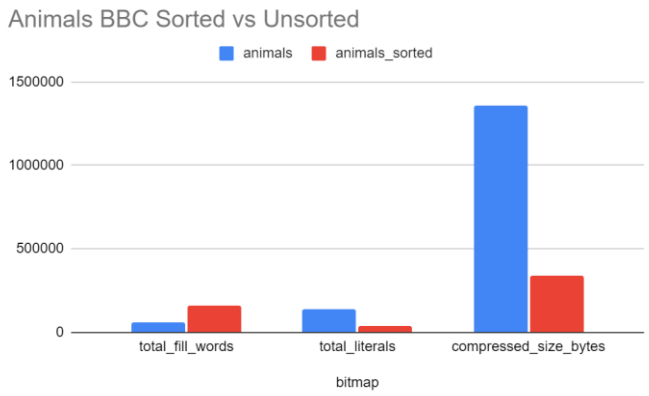
The word compression rates were for cases on the unsorted bitmap using WAH with a wordcount 16 or higher. This resulted in compressed bitmaps of about 1600000 which is almost no decrease.

Effects of Sorting



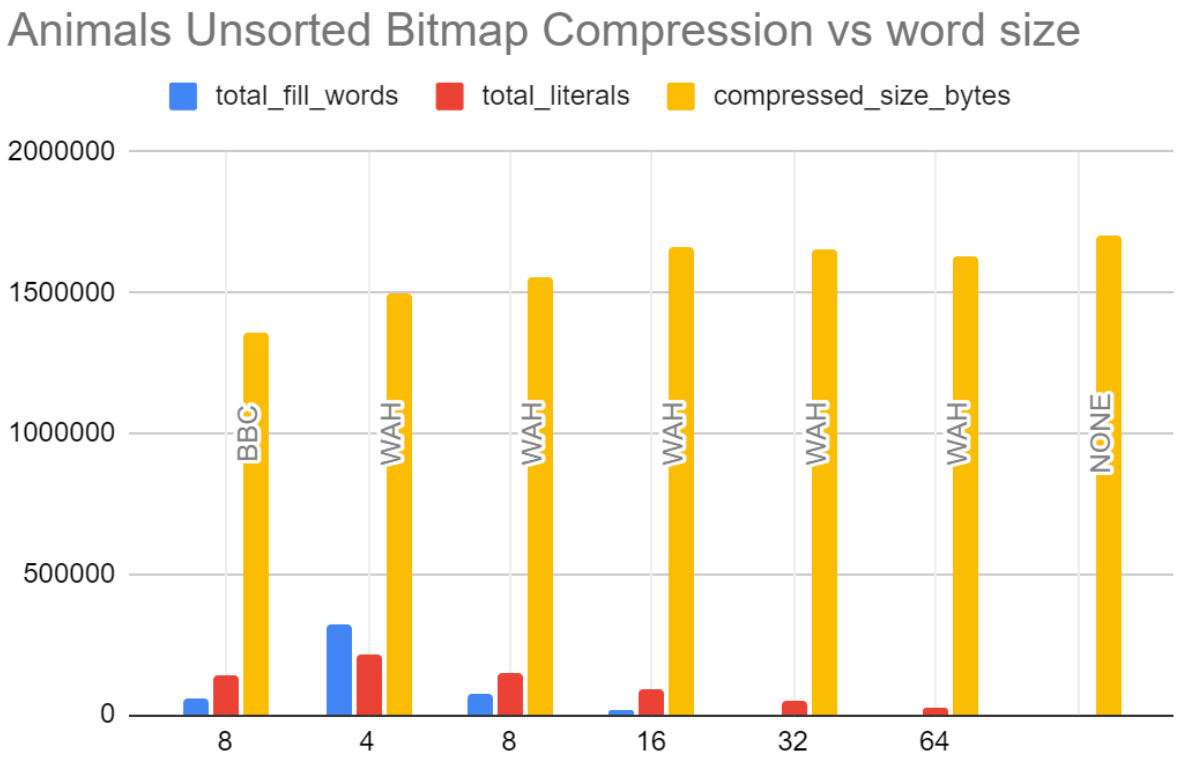
Sorting significantly reduces the compressed size for both BBC and WAH compression methods across all word sizes. For instance, in the case of BBC compression, sorting reduces the average compressed size by four times. Similarly, for WAH compression with an 8-bit word size, sorting reduces the size from 1,558,008 bytes to just 51,744 bytes.

The dramatic reduction in compressed size with sorting is attributed to the increase in the fill to literal ratio. Sorting tends to group identical bits together, increasing the number of fill words.



This is evident in the data; for example, in BBC compression (8-bit word size), the ratio increases from 0.42 (unsorted) to 4.22 (sorted), indicating that sorting leads to more efficient compression by creating longer runs of identical bits.

Effects of Different Word Sizes on WAH



Different word sizes yield different compression efficiencies. For the unsorted bitmap a word size of 4 had the highest compression rate and fill to literal ratio. Overall increasing the word size does not significantly change the compressed size.

For sorted bitmaps, the effect is significant. The best word size for the sorted animals bitmap was 8, and the worst was 4. This is because with a word size of 4 there is not much room in the header bytes to store the number of runs, and with a sorted bitmap there are exceptionally large numbers of runs.

Larger word sizes do not always result in a better compression ratio, the best word size will depend on the bitmap.

BBC vs WAH

From my results, my conclusion was that BBC performed better when dealing with an unsorted bitmap, and WAH performed better when dealing with a sorted bitmap. I infer this is because BBC offers only one-sided compression while with WAH you can store runs of 1's and 0's.