Profiling

Caden Corontzos

I used callgrind to profile my code. Callgrind finds out how many instructions are associated with each line of code, which gives an indication of how long each line will take. I looked at some of the trouble spots that Eitan and I found to see if we could see a noticable difference in runtime before and after each change.

I used the two groups of DNA I had found the other day as benchmarks.

Here is the data before any optimizations.

Table 1: Corpus 1

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus1/humprtb	56737	21902	2.590	7.2	1.8
DNACorpus1/humdyst	38770	15300	2.534	5.0	1.0
DNACorpus1/vaccg	191737	70067	2.736	17.2	7.0
DNACorpus1/hehcmv	229354	85526	2.682	22.8	8.6
DNACorpus1/mpomtcg	186609	70254	2.656	19.6	7.0
DNACorpus1/humhdab	58864	22699	2.593	7.8	2.0
DNACorpus1/chmpxx	121024	43516	2.781	11.8	4.0
DNACorpus1/mtpacga	100314	36862	2.721	11.0	3.8
DNACorpus1/chntxx	155844	58336	2.671	16.4	5.8
DNACorpus1/humghcs	66495	25552	2.602	7.6	2.4
DNACorpus1/humhbb	73308	28134	2.606	8.6	3.0

Table 2: Corpus 2

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus2/YeMi	73689	27235	2.706	8.6	3.0
DNACorpus2/DaRe	62565020	19586457	3.194	14945.2	1888.0
DNACorpus2/EnIn	26403087	8609993	3.067	5396.8	764.6
DNACorpus2/HePy	1667825	566972	2.942	146.4	38.4
DNACorpus2/OrSa	43262523	14148071	3.058	9803.2	1343.8
DNACorpus2/EsCo	4641652	1593404	2.913	564.6	121.2
DNACorpus2/GaGa	148532294	46851765	3.170	40917.6	4781.4
DNACorpus2/WaMe	9144432	3112000	2.938	1445.2	253.0
DNACorpus2/ScPo	10652155	3590856	2.966	1742.0	291.8
DNACorpus2/AnCa	142189675	43665091	3.256	38822.4	4397.6
DNACorpus2/HaHi	3890005	1306708	2.977	445.6	98.0
DNACorpus2/HoSa	189752667	57200209	3.317	53534.6	5877.0
DNACorpus2/AeCa	1591049	556535	2.859	141.8	39.6
DNACorpus2/DrMe	32181429	10619042	3.031	7174.0	1005.2
DNACorpus2/BuEb	18940	7893	2.400	2.0	0.0
DNACorpus2/PlFa	8986712	2895744	3.103	1364.8	236.4
DNACorpus2/AgPh	43970	17442	2.521	6.0	1.8

I then tested my implementation with callgrind. I encoded HaHi from DNA Corpus 2 to see what lines are taking long.

The first change we want to make was to create a type for codewords. They were previously just int, but we want to make them fixed at uint64 and have a special type for them.

I ran callgrind before and after making this change. I found that the amount of instructions was virtually identical, which makes sense since we didn't really change the structure of the code. I think that having a fixed codeword size will show some time saved once we turn compiler optimizations back on, but without that it didn't change much. I rand the corpus again just to confirm.

Table 3: Corpus 1

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus1/humprtb	56737	21902	2.590	7.2	2.0
DNACorpus1/humdyst	38770	15300	2.534	5.0	1.2
DNACorpus1/vaccg	191737	70067	2.736	18.6	7.0
DNACorpus1/hehcmv	229354	85526	2.682	24.2	8.4
DNACorpus1/mpomtcg	186609	70254	2.656	19.4	7.0
DNACorpus1/humhdab	58864	22699	2.593	7.8	2.0
DNACorpus1/chmpxx	121024	43516	2.781	12.8	4.0
DNACorpus1/mtpacga	100314	36862	2.721	11.2	3.8
DNACorpus1/chntxx	155844	58336	2.671	15.8	5.6
DNACorpus1/humghcs	66495	25552	2.602	8.0	2.2
DNACorpus1/humhbb	73308	28134	2.606	8.8	2.8

Table 4: Corpus 2

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus2/YeMi	73689	27235	2.706	8.6	3.0
DNACorpus2/DaRe	62565020	19586457	3.194	14928.6	1871.2
DNACorpus2/EnIn	26403087	8609993	3.067	5390.4	766.0
DNACorpus2/HePy	1667825	566972	2.942	143.0	40.4
DNACorpus2/OrSa	43262523	14148071	3.058	9791.2	1337.4
DNACorpus2/EsCo	4641652	1593404	2.913	560.4	120.4
DNACorpus2/GaGa	148532294	46851765	3.170	40835.4	4727.4
DNACorpus2/WaMe	9144432	3112000	2.938	1444.8	250.8
DNACorpus2/ScPo	10652155	3590856	2.966	1744.6	291.4
DNACorpus2/AnCa	142189675	43665091	3.256	38805.8	4381.2
DNACorpus2/HaHi	3890005	1306708	2.977	443.6	97.8
DNACorpus2/HoSa	189752667	57200209	3.317	53561.4	5840.4
DNACorpus2/AeCa	1591049	556535	2.859	140.8	39.8
DNACorpus2/DrMe	32181429	10619042	3.031	7158.8	1000.6
DNACorpus2/BuEb	18940	7893	2.400	2.0	0.2
DNACorpus2/PlFa	8986712	2895744	3.103	1361.6	236.4
DNACorpus2/AgPh	43970	17442	2.521	6.0	2.0

These times look pretty similar to the first run.

One thing I noticed in the callgrind output was that there was significant time being spent each time we check if something is in the dictionary.

Here is the callgrind output for that line.

```
105,030,135 ( 0.18%)          if (dictionary.find(currentBlock + next_character) != dictionary.end()){
13,537,450,317 (22.83%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std
11,653,779,430 (19.65%)  => /usr/include/c++/9/bits/unordered_map.h:std::unordered_map<std::__cxx11::ba
2,108,383,120 ( 3.56%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std:
956,941,242 ( 1.61%)  => /usr/include/c++/9/bits/unordered_map.h:std::unordered_map<std::__cxx11::basic
241,180,314 ( 0.41%)  => /usr/include/c++/9/bits/hashtable_policy.h:bool std::__detail::operator!=<std:
```

As shown, this line is taking a significant amount of instructions, and it needs to pull the end() of the dictionary each time it is ran. If we use cend() instead and save that iterator in a variable called end, we can save a significant amount of instructions.

```
89,470,115 ( 0.61%)          if (dictionary.find(currentBlock + next_character) != end ){
3,353,009,053 (22.78%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std:
2,833,786,025 (19.26%)  => /usr/include/c++/9/bits/unordered_map.h:std::unordered_map<std::__cxx11::bas
420,120,704 ( 2.85%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std::c
50,570,065 ( 0.34%)  => /usr/include/c++/9/bits/hashtable_policy.h:bool std::__detail::operator!=<std::
```

As you can see, the line itself takes less instructions, and it no has to pull up the end of the dictionary every time. This significantly decreased the total instructions in the program run. I then ran the corpus again to check the times.

Table 5: Corpus 1

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus1/humprtb	56737	21902	2.590	7.0	2.0
DNACorpus1/humdyst	38770	15300	2.534	4.4	1.0
DNACorpus1/vaccg	191737	70067	2.736	18.6	6.4
DNACorpus1/hehcmv	229354	85526	2.682	21.4	9.0
DNACorpus1/mpomtcg	186609	70254	2.656	19.4	6.8
DNACorpus1/humhdab	58864	22699	2.593	6.6	2.0
DNACorpus1/chmpxx	121024	43516	2.781	12.2	4.0
DNACorpus1/mtpacga	100314	36862	2.721	10.2	3.8
DNACorpus1/chntxx	155844	58336	2.671	15.6	6.0
DNACorpus1/humghcs	66495	25552	2.602	8.6	2.4
DNACorpus1/humhbb	73308	28134	2.606	9.2	2.8

Table 6: Corpus 2

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus2/YeMi	73689	27235	2.706	8.4	3.0
DNACorpus2/DaRe	62565020	19586457	3.194	14962.6	1879.2
DNACorpus2/EnIn	26403087	8609993	3.067	5402.2	762.6
DNACorpus2/HePy	1667825	566972	2.942	145.6	39.4
DNACorpus2/OrSa	43262523	14148071	3.058	9797.0	1331.0
DNACorpus2/EsCo	4641652	1593404	2.913	560.0	120.8
DNACorpus2/GaGa	148532294	46851765	3.170	40689.8	4761.2
DNACorpus2/WaMe	9144432	3112000	2.938	1441.8	249.4
DNACorpus2/ScPo	10652155	3590856	2.966	1741.6	289.6
DNACorpus2/AnCa	142189675	43665091	3.256	38776.8	4389.6
DNACorpus2/HaHi	3890005	1306708	2.977	444.6	97.8
DNACorpus2/HoSa	189752667	57200209	3.317	53730.6	5872.4
DNACorpus2/AeCa	1591049	556535	2.859	141.2	40.4
DNACorpus2/DrMe	32181429	10619042	3.031	7161.0	999.4
DNACorpus2/BuEb	18940	7893	2.400	2.0	0.2
DNACorpus2/PlFa	8986712	2895744	3.103	1359.4	233.8
DNACorpus2/AgPh	43970	17442	2.521	5.6	1.8

It's hard to see in the smaller files but if you look at the larger files in Corpus 2 you can see that the runtime has been reduced.

Another thing that I noticed from the callgrind output was that a lot of time/instructions are being spent on string concatenation. Every time we have already seen a sequence, we have to concatenate a character. I also noticed that I was doing this concatenation multiple times without needing to.

```
        if (dictionary.find(currentBlock + next_character) != end ){
            currentBlock = currentBlock + next_character;
        }
        else{

            // other code here ommitted

            dictionary[currentBlock + next_character] = codeword;
        }
```

If I just concatenate them and save the output into a new string, that will save me from doing the concatenation 2 more times.

After another run of the corpuses

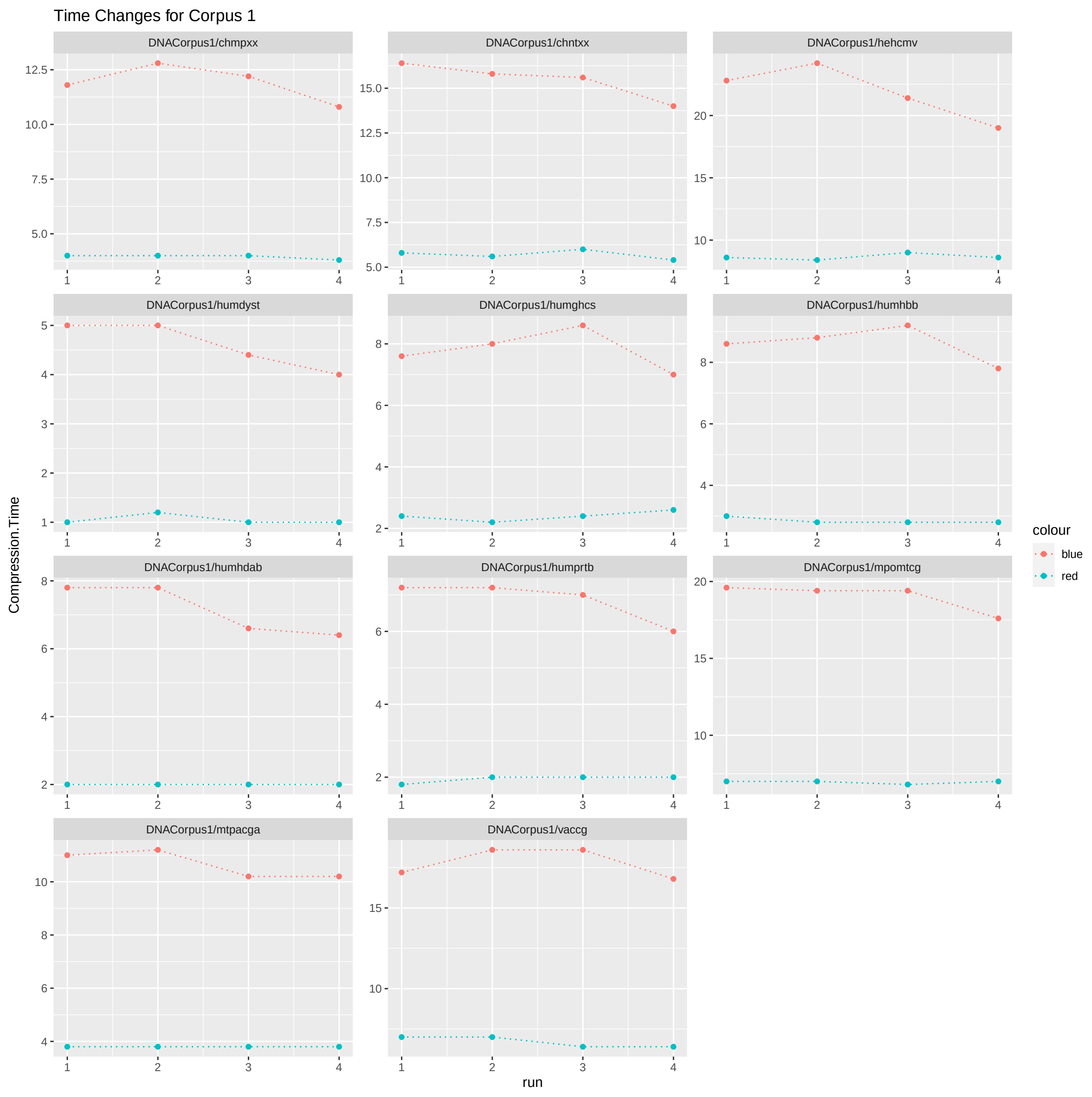
Table 7: Corpus 1

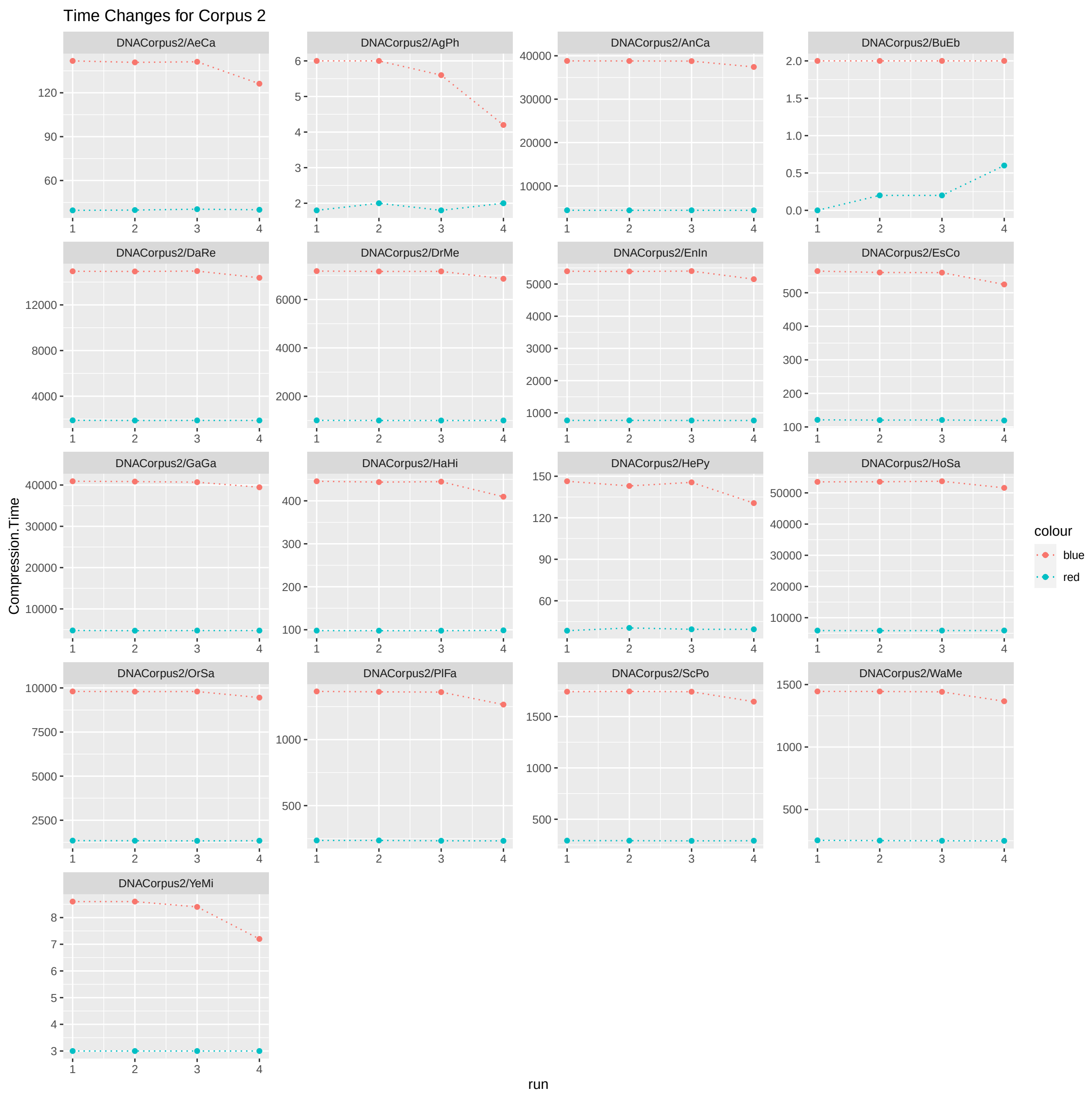
File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus1/humprtb	56737	21902	2.590	6.0	2.0
DNACorpus1/humdyst	38770	15300	2.534	4.0	1.0
DNACorpus1/vaccg	191737	70067	2.736	16.8	6.4
DNACorpus1/hehcmv	229354	85526	2.682	19.0	8.6
DNACorpus1/mpomtcg	186609	70254	2.656	17.6	7.0
DNACorpus1/humhdab	58864	22699	2.593	6.4	2.0
DNACorpus1/chmpxx	121024	43516	2.781	10.8	3.8
DNACorpus1/mtpacga	100314	36862	2.721	10.2	3.8
DNACorpus1/chntxx	155844	58336	2.671	14.0	5.4
DNACorpus1/humghcs	66495	25552	2.602	7.0	2.6
DNACorpus1/humhbb	73308	28134	2.606	7.8	2.8

Table 8: Corpus 2

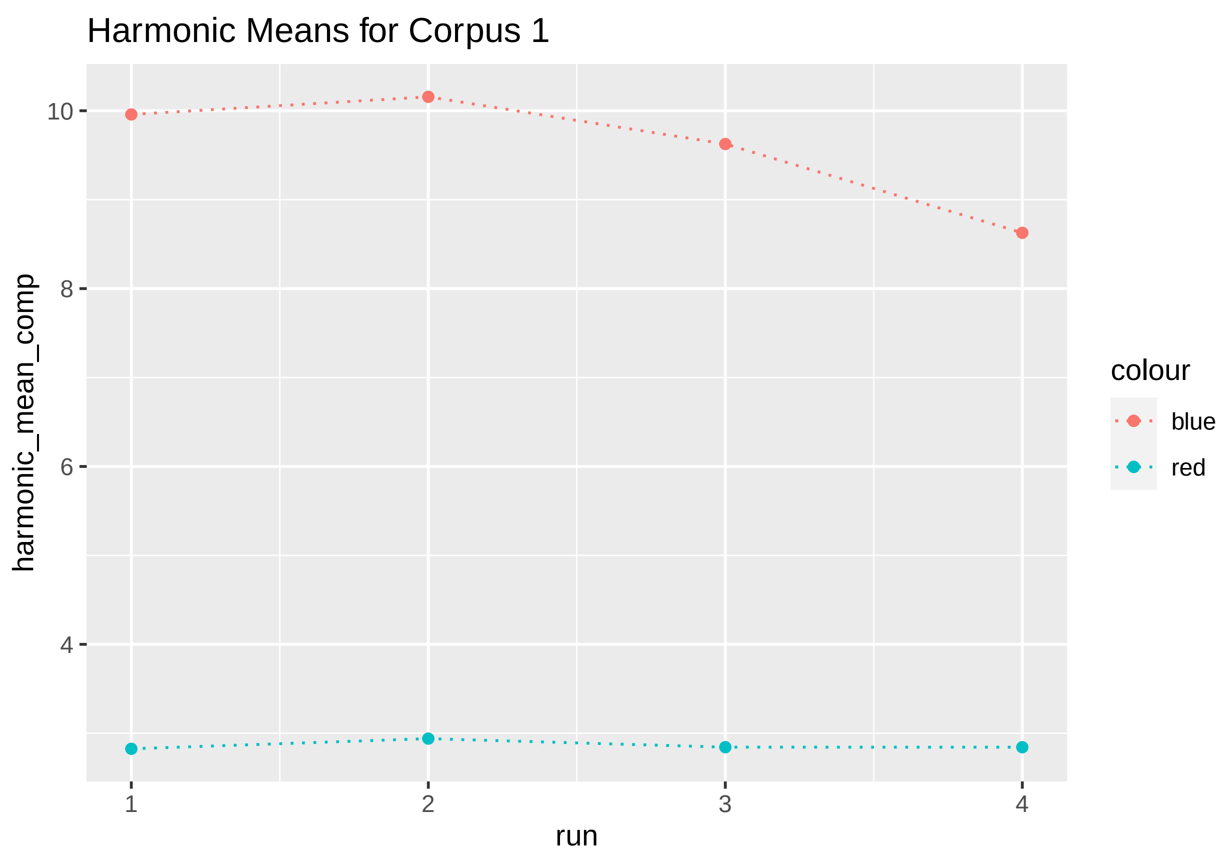
File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compression.Time	Decompression.Time
DNACorpus2/YeMi	73689	27235	2.706	7.2	3.0
DNACorpus2/DaRe	62565020	19586457	3.194	14374.2	1876.8
DNACorpus2/EnIn	26403087	8609993	3.067	5150.4	761.8
DNACorpus2/HePy	1667825	566972	2.942	130.6	39.4
DNACorpus2/OrSa	43262523	14148071	3.058	9452.0	1338.0
DNACorpus2/EsCo	4641652	1593404	2.913	524.8	119.2
DNACorpus2/GaGa	148532294	46851765	3.170	39455.0	4756.2
DNACorpus2/WaMe	9144432	3112000	2.938	1366.4	248.8
DNACorpus2/ScPo	10652155	3590856	2.966	1644.8	290.8
DNACorpus2/AnCa	142189675	43665091	3.256	37427.4	4375.2
DNACorpus2/HaHi	3890005	1306708	2.977	409.6	98.6
DNACorpus2/HoSa	189752667	57200209	3.317	51606.8	5889.4
DNACorpus2/AeCa	1591049	556535	2.859	126.2	40.0
DNACorpus2/DrMe	32181429	10619042	3.031	6855.6	1001.4
DNACorpus2/BuEb	18940	7893	2.400	2.0	0.6
DNACorpus2/PlFa	8986712	2895744	3.103	1265.2	233.2
DNACorpus2/AgPh	43970	17442	2.521	4.2	2.0

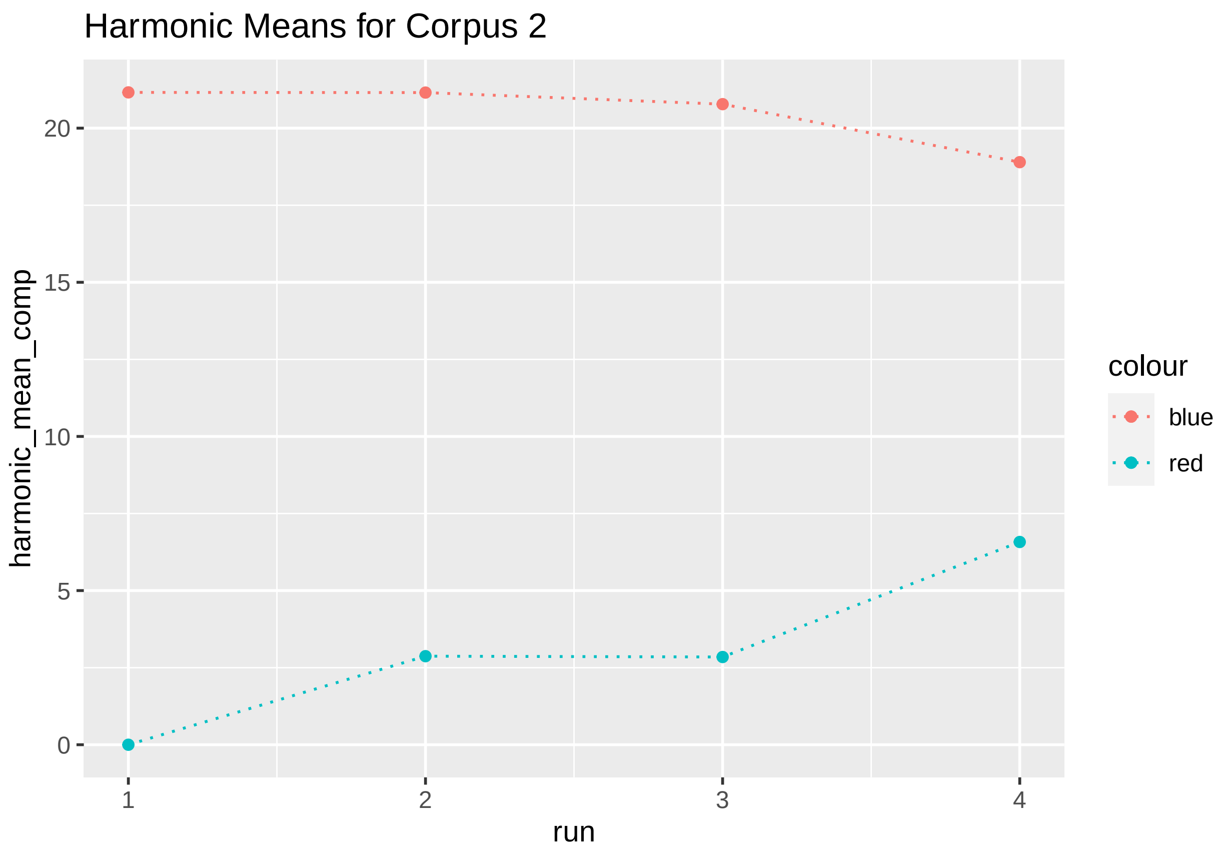
As we can see, the compression is a good bit faster, especially for the larger files. Let's look at the time improvement over the course of these changes.


Time Changes for Corpus 1
DNACorpus1/chmpxx
DNACorpus1/chntxx
DNACorpus1/hehcmv
DNACorpus1/humdyst
DNACorpus1/humghcs
DNACorpus1/humhbb
DNACorpus1/humhdab
DNACorpus1/humprtb
DNACorpus1/mpomtcg
DNACorpus1/mtpacga
DNACorpus1/vaccg
Compression.Time
run
colour
blue
red



and the harmonic averages


Harmonic Means for Corpus 1
harmonic_mean_comp
10
8
6
4
1
2
3
4
run
colour
blue
red



Harmonic Means for Corpus 2
harmonic_mean_comp
run
colour
blue
red
20
15
10
5
0
1
2
3
4

"‘