Aproximate counting with a Floating-Point Counter

João Tiago Lacerda Rainho 92984

Abstract **– When counting very frequent events such as Wi-fi packets or DNA sequences, the memory space becomes an issue, for that reason multiple techniques were found to help in this problem such as the Fixed Probability Counter and the Floating-Point Counter. The objective is not to count every event but to estimate the real count with an approximation that is much more space-efficient.**

Keywords **– Approximate Floating-point Csurös Counter**

I. Introduction

Counting events is an important subject to study because when analysing complex systems, the same event might happen multiple times to an extent, which can create memory issues. As an example when counting a type of packet passing a router or how many times a molecule interacts with another molecule on a large simulation. This problem has a lot of solutions such as only counting a fraction of times the event happens (approximate counter with fixed probability) which is better than just count every event, however this is still linearly increasing. Other techniques such as Csurös Counter (approximate counter with decreasing probability) are far better because even when highly frequent events happen the counter will not be linear but logarithmic. The last one will be the main subject of this paper.

II. Methods

A. Counter Representation

A Counter is represented by:

- **X**: number of times an event got incremented

The code is structured to be easy to implement new implementations of **Counters**. The default counter (**Counter**) is an exact counter as seen in the following code:

```
class Counter:
    x:int

    def __init__(self)->None:
        self.x = 0

    def increment(self)->None:
        self.x += 1

    @property
    def value(self)->int:
        return self.x
```

In order to create a custom counter implementation, it is only needed to create a new class, extend the previously mentioned counter and override the needed methods (*init*, *increment* and *value*).

```
class CustomCounter(Counter):

    def __init__(self)->None:
        super().__init__()

    def increment(self)->None:
        pass

    @property
    def value(self)->int:
        pass
```

Other counter implementations were also implemented such as:

1. Fixed Probability Counter
2. Floating-Point Counter (Csurös Counter)

The **Fixed Probability Counter** is characterized by only incrementing when the generated random value is lower than the assigned probability to increment. When estimating the counter it is the inverse of that probability times the number of increments performed.

$$\frac{1}{prob} \times X$$

Although is more efficient than the exact counter, this counter has obvious disadvantages such as in low frequent events the error will be very large.

The **Floating-Point Counter** as said before is an approximate counter with decreasing probability. This means it is more efficient in memory than the previous as it only increments the value based on how many increments have already done and with larger values it increments less often. It is also more precise in small countings because for each d increments there is a exponentially decreasing probability of incrementing the stored value being the first d steps with 100% probability and therefore performing exact counting. In order to estimate the count value of this counter:

$$(M + X \bmod M) \times 2^{\lfloor \frac{X}{M} \rfloor} - M$$

B. Floating-Point Counter Performance Improvement

In other implementations of a **Floating-Point Counter** the increment method would be:

```
def increment(self)->None:
    t:int = floor(self.x/self.m)
    while t > 0:
        if randint(0,1) == 1: return
        t -= 1
    super().increment()
```

This is not so optimized for very frequent events because when t increases, then there are constantly two

comparisons for each decrement on t, this means a lot of lost performance when we reach the point the t is already beyond 10. Since the main delay of this function is that *while* followed by an *if* and all it does is to multiply probabilities as the following:

$$\prod_{0}^{t} \frac{1}{2}$$

(eg: in first loop the prob is 50%, then in the second is 25% because its 50% of the other 50%, then in the third 12.5% and so on).

We can just compute the whole probability with the t and with the probability of every step (50%) in order to get the same result faster.

$$\prod_{0}^{t} \frac{1}{2} = \frac{1}{2}^{t}$$

On a counter to 10 million this is responsible for a drop in processing time from 11.41 seconds to 2.13 seconds which is a decrease of 435.68%, it is also noticeable that the larger the count the more time it will be saved from this implementation:

```
def increment(self)->None:
    t:int = floor(self.x/self.m)
    prob:float = (0.5)**(t)
    if random() < prob:
        super().increment()
```

C. Error calculation

In order to compare different counters there needs to be a way of quantify how accurate some counters are in relation to others, for that reason multiple error measurements are taken:

1. Accuracy
2. Mean Absolute Error (MAE)
3. Mean Relative Error (MRE)
4. Max Deviation (MD)
5. Mean Absolute Deviation (MAD)
6. Maximum value
7. Minimum value
8. Bits

Note: All comparisons are made from a list of estimated results and the truth

The **Maximum** and **Minimum values of each**

The **Accuracy** is calculated by

$$Accuracy = (1 - \mid (\frac{1}{n}\sum_{i=1}^{n}\frac{estimated\_value_i}{truth}) - 1 \mid) \times 100$$

The **Mean Absolute Error** is calculated by

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid estimated\_value_i - truth \mid$$

The **Mean Relative Error** is calculated by

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\frac{\mid estimated\_value_i - truth \mid}{truth}$$

The **Mean Absolute Deviation** is calculated by

$$MAD = \frac{1}{n}\sum_{i=0}^{n} \mid estimated\_value_i - \frac{1}{n}\sum_{j=0}^{n} estimated\_value_j \mid$$

The **Maximum Deviation** is calculated by

$$MD = \max(\mid v - \frac{1}{n}\sum_{j=0}^{n} estimated\_value_j \mid)$$

The **Bits** are calculated by checking the binary value for the first "1" counting from the left because we assume the counts are always positive so we don't need to worry about two's complement representations because it will be normal binary.

$$Bits = \frac{1}{n}\sum_{i=0}^{n} length(binary(value_i)) - IndexFirstOne(binary(value_i))$$

D. Auxiliary Features

Some features were implemented in order to improve result extraction (**results.py**) and result analysis (**visualizer.py**).

The results.py is optimized to read large texts and return the statistics of the counters involved. It's optimization is done by only reading once the file (io intensive task), this means that after reading the file, all operations are more efficient by only using the Computer Processing Unit (CPU) and not waiting for the Bus transfers (information highway of the computer). This also saves memory (RAM) by not needing to store all the text in memory. In order to maintain all important information by only reading once, a dictionary containing the char as key and an exact counter as value is used. This provides easy and fast testing after by only incrementing the testing counters the number of times the exact counter was incremented which we have direct access.

The visualizer.py is used to display multiple charts which visually demonstrates the differences between the counters. The characters are always in the same position and in decreasing order from the left to the right in order to provide the readers a stable analysing when comparing the same characters. (Enlarged image on 3)

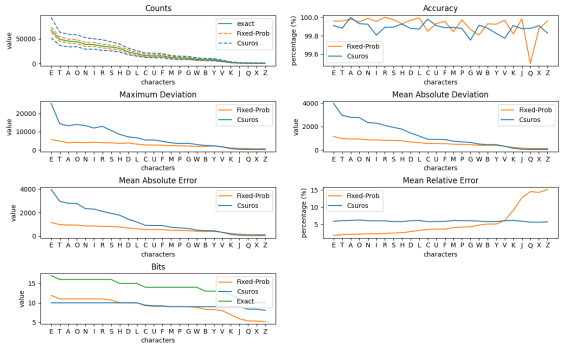

Fig. 1
Comparison Visualization of Fixed Probability Counter with p=1/32 and Floating-Point Counter with d=6

III. Experimental Results and Discussion

All results are calculated using $seed = 100$ and $number\_of\_trials = 5000$

A. Figure 2 Analysis

In the Figure 2, we have the counting of the characters of the book in analysis 5000 times with a Fixed Probability Counter (FPC) with $P = \frac{1}{64}$ and a Floating-Point Counter, also known as Csurös Counter (CC) with $d = 2$. This example was done to demonstrate **badly chosen parameters** for both the counters. The FPC has a probability of incrementation too small which means that the error for low frequent events will be larger. The CC has a step size (d) too low, this means that in just a few increments, the mean absolute error will increase. The relative error maintains constant because although the error increases, the number of increments also increases at approximately the same rate.

As seen in the Counts chart, represented as green is the Exact Counter. For every other Counter, based on the fact that they are not deterministic but probabilistic, there will be two dotted lines with their respective color, the lower line represents the minimum value and the higher represents the maximum value. Starting with the FPC, as the counter is incremented, the margin from the exact counter to the FPC margin is linearly increasing, however this happens very slowly because the probability of incrementing is constant. With regard to CC, the margins gets wider the higher the counter, because the probability of incrementing is exponentially decreasing. The d is so low, then every probabilistic update is faster to go though, this allows lucky strikes which overestimate the exact counter. The opposite is also possible, misfortune will lead to underestimate the real value. The wide values are also fruit of the enormous amount of trials done, if instead of 5000 trials we did 100, the width of that gap would be much lower.

In the Accuracy chart, we can see that the FPC is more constant for larger counts since the MRE is lower the larger the count is. When smaller counts are being treated, the MRE is higher so the accuracy will be lower. In the CC the accuracy will be more unstable, however remains in the same range of values because the MRE is also constant.

The deviation charts are easily explained with the already said. The FPC has a decreasing MRE, and because of that the MAD will remain almost constant, only increasing by the multiplication factor of the number of counted characters. However in the CC, the MRE is constant so the MAD will be linearly increasing as the number of counted characters also increases.

To finalise this subsection, the space efficiency is very good to the CC because it takes so little space compared to the others as it follows a logarithmic function, median (about in the middle from the CC and the Exact Counter) to the FPC and much higher for the Exact Counter (which is worse) as expected. Both the FPC and the Exact Counter have approximately the same behaviour, the FPC is simply lower based on its parameter p.

B. Figure 3 Analysis

In the Figure 3 we have **better parameters**, which would be useful to use in practical applications with huge counts. The parameters are $p = \frac{1}{32}$ for the FPC and $d = 12$ for the CC. These values provide both the counters a way of minimizing their weaknesses. For the FPC increases the probability of incrementing low frequent counts, on the other hand, the CC increases the amount of steps to switch to the next probability to increment, which will provide more stable results as the MRE remains consistently very low. Comparatively with the Figure 2, the MRE is lower for the FPC however the trend is similar which is explained by the still constant probability but higher (twice as high). The CC, as said before, remains reasonably consistent but on lower percentages which is a good thing.

On the Counts chart, the margin from the exact counter to the minimum and maximum values are very near to the exact count, this means that both the counters are doing a good job at approximating the real value, the CC even outperforms the FPC. We can see a huge difference compared to the Figure 2 which has very wide margins especially on the CC which is a consequence of a superior error which intern increases the MD.

In the Accuracy chart, the CC remains very high reaching almost 100% accuracy while the FPC although converges to a good percentage on the high frequency events, on the lower frequency events, still reaches a deficit of about 0.5%. This chart is much better than the original (Figure 2) as the inaccuracies are almost non existent.

The MAD chart also shows a more horizontal line compared to the same chart on the Figure 2 which is explained by the lower MRE on both counters, more noticeable the CC.

Finally, the number of bits needed to use in each implementation is almost the same in both the exact counter and in the CC, being the CC a bit lower for this example, however for large enough counts, the FPC will reach and overtake the CC. The FPC is the most space efficient in this case. In order to improve this, the parameter d must be lower.

C. Figure 4 Analysis

The last analysis is using FPC with $p = 1/32$ and CC with $d = 6$. This parameters were found to be a better match for this specific project which might have large counts but not huge. We can see that while the accuracy remains high, the number of bits used are much more consistent and even lower for the CC which means that both the space-complexity and the overall accuracy are better (for both small and large counts). The MRE is constant hovering the 5-6% and the accuracy always above 99.8%.

IV. Conclusion

In conclusion, the CC with a balanced d value can achieve a very good, almost optimal, overall accuracy especially in the extremes (both higher and lower frequent events) where other techniques, such as FPC, struggled. This is achieved by decreasing the MRE which is a consequence of increasing the size of each probabilistic update to an optimal value different for each application. On our case the $d = 6$ was found be very reasonable surpassing the FPC in the most relevant benchmarks which are the overall accuracy and the necessary number of bits to represent the count.

References

M. Csurös. Approximate counting with a floating-point counter. In COCOON, LNCS vol. 6196, p. 358-367, Springer, 2010

Appendix

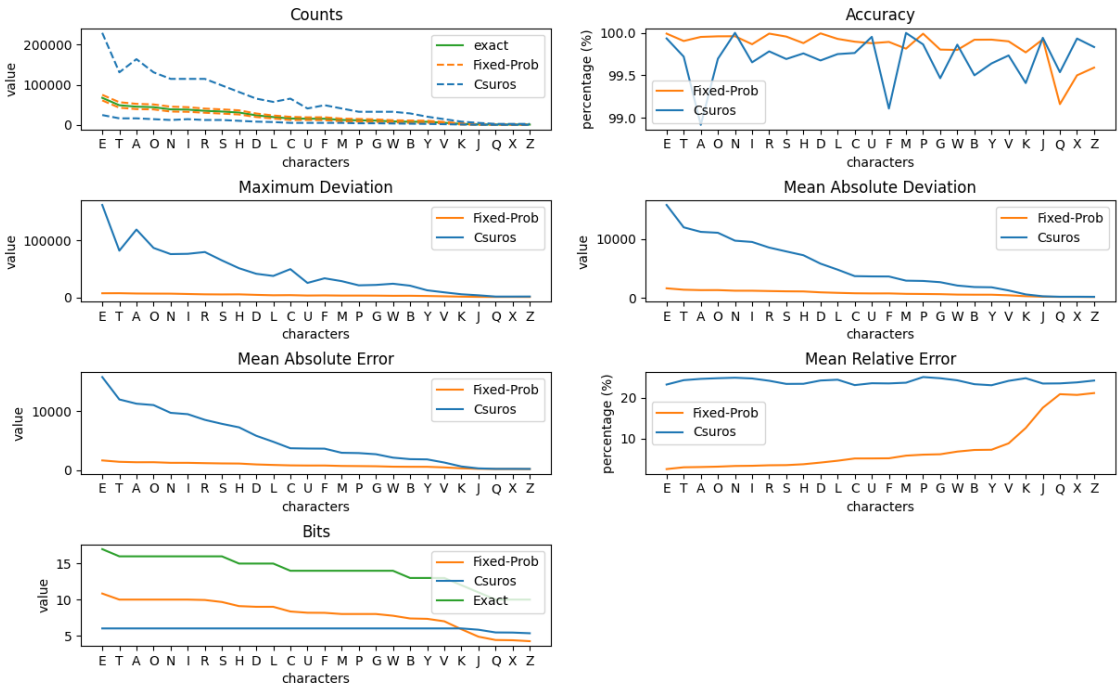

Fig. 2
Comparison Visualization of Fixed Probability Counter with p=1/64 and Floating-Point Counter with d=2

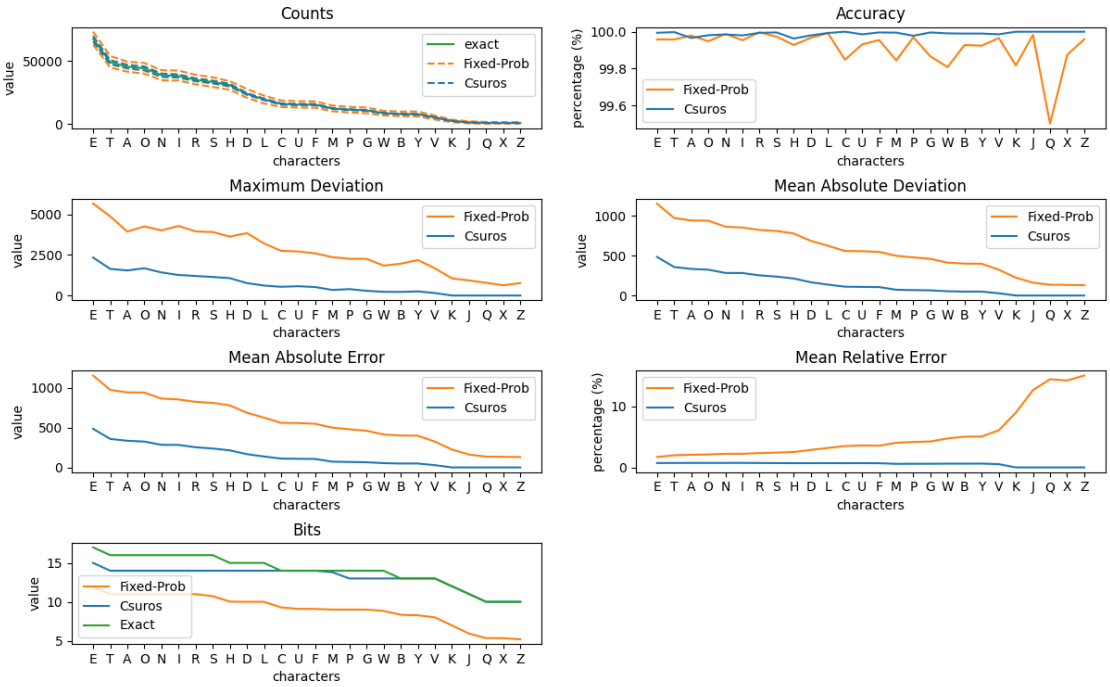

Fig. 3

Comparison Visualization of Fixed Probability Counter with p=1/32 and Floating-Point Counter with d=12

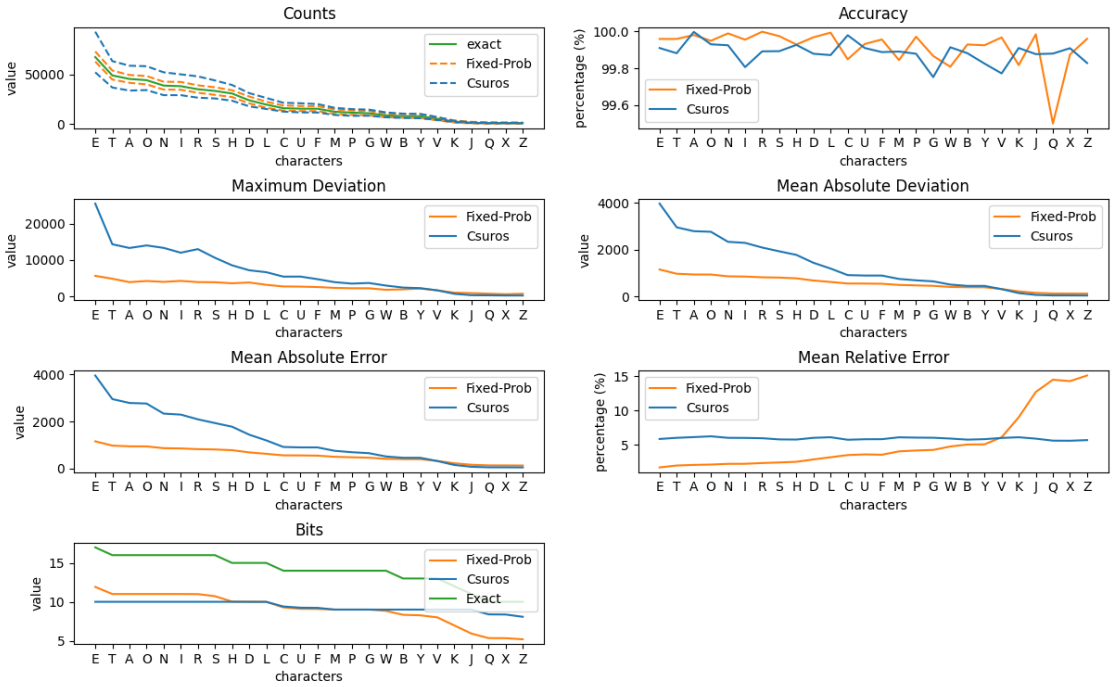

Fig. 4

Comparison Visualization of Fixed Probability Counter with p=1/32 and Floating-Point Counter with d=6

Character	Counter	Accuracy	MD	MAD	MAE	MRE	MAX	MIN	BITS
E	FPC	99.99	7411.469	1646.422	1646.512	2.438	74944	61056	10.839
	CC	99.933	161878.016	15724.86	15742.272	23.308	229372	24572	6.0
T	FPC	99.903	7513.434	1411.081	1411.11	2.879	56576	42880	10.0
	CC	99.719	81915.085	11943.823	11954.48	24.389	131068	16380	6.0
A	FPC	99.951	6975.578	1339.669	1339.819	2.946	52480	39424	10.0
	CC	98.916	118847.078	11178.632	11237.604	24.708	163836	16380	6.0
O	FPC	99.958	6812.429	1344.075	1343.949	3.038	51072	38656	10.0
	CC	99.694	86691.43	11024.423	11003.22	24.871	131068	14332	6.0
N	FPC	99.959	6718.067	1241.534	1241.63	3.203	45504	33344	10.0
	CC	99.999	75913.626	9689.905	9689.895	24.993	114684	12284	6.0
I	FPC	99.865	6120.397	1238.11	1239.622	3.249	44224	32704	9.999
	CC	99.652	76396.134	9474.056	9464.756	24.806	114684	14332	6.0
R	FPC	99.99	5553.485	1184.564	1184.489	3.376	40640	30272	9.945
	CC	99.781	79524.25	8525.848	8504.328	24.241	114684	12284	6.0
S	FPC	99.955	5366.963	1139.154	1139.459	3.415	38720	28224	9.665
	CC	99.691	64829.03	7871.661	7830.144	23.466	98300	12284	6.0
H	FPC	99.878	5501.338	1119.097	1119.363	3.633	36352	25984	9.095
	CC	99.757	51177.882	7231.984	7237.825	23.49	81916	10236	6.0
D	FPC	99.993	4613.709	967.703	967.658	4.043	28544	19712	9.0
	CC	99.674	41521.971	5811.841	5816.678	24.305	65532	8188	6.0
L	FPC	99.928	3905.088	881.235	881.653	4.499	23488	15936	8.999
	CC	99.749	37693.85	4800.527	4801.353	24.5,	57340	7164	6.0
C	FPC	99.895	4129.741	807.861	807.847	5.054	20096	12224	8.349
	CC	99.762	49586.995	3704.14	3703.092	23.169	65532	5116	6.0
U	FPC	99.877	3431.066	783.324,	783.629	5.066	18880	12224	8.177
	CC	99.952	25480.602	3655.669	3655.174	23.631	40956	5116	6.0
F	FPC	99.892	3669.67	786.31	786.217	5.099	19072	11840	8.164
	CC	99.108	33591.501	3643.622	3636.582	23.585	49148	5116	6.0
M	FPC	99.813	3357.862	711.549	711.438	5.755	15744	9088	9.088
	CC	99.998	28592.742	2937.935	2937.86	23.763	40956	5116	6.0
P	FPC	99.989	3367.718	688.374	688.362	5.997	14848	8576	8.0
	CC	99.863	21269.299	2889.056	2888.032	25.159	32764	4092	6.0
G	FPC	99.802	3265.587	660.275	660.618	6.095	13824	7552	7.999
	CC	99.465	21983.027	2681.644	2694.967	24.864	32764	4092	6.0
W	FPC	99.798	2968.538	587.389	587.474	6.755	11648	6400	7.762
	CC	99.86	24079.155	2113.812	2117.697	24.35	32764	3580	6.0
B	FPC	99.918	3040.486	569.831	569.672	7.156	11008	5120	7.382
	CC	99.499	20667.085	1861.183	1862.333	23.393	28668	3068	6.0
Y	FPC	99.919	2636.352	567.316	567.207	7.211	10496	5568	7.331
	CC	99.639	12581.581	1820.326	1820.065	23.138	20476	2556	6.0
V	FPC	99.899	2091.622	468.363	468.137	8.788	7424	3584	6.988
	CC	99.733	9019.238	1286.372	1290.194	24.22	14332	1788	6.0
K	FPC	99.769	1475.763	314.013	314.067	12.573	3968	1024	5.888
	CC	99.408	5675.213	620.454	621.265	24.87	8188	1020	6.0
J	FPC	99.922	1093.99	224.238	224.194	17.584	2368	256	4.854
	CC	99.941	3841.754	300.454	300.366	23.558	5116	444	5.820
Q	FPC	99.16	911.13	195.845	196.088	20.927	1856	256	4.393
	CC	99.536	1614.656	221.588	221.067	23.593	2556	316	5.44
X	FPC	99.499	988.685	194.102	194.195	20.747	1920	128	4.359
	CC	99.932	1620.634	223.082	223.178	23.844	2556	316	5.424
Z	FPC	99.59	930.547	183.294	183.422	21.205	1792	192	4.229
	CC	99.832	1692.454	210.18	210.172	24.297	2556	252	5.329

TABLE I

DISCRETE COMPARISON OF FIXED PROBABILITY COUNTER WITH P=1/64 AND FLOATING-POINT COUNTER WITH D=2

Character	Counters	Accuracy	MD	MAD	MAE	MRE	MAX	MIN	BITS
E	FPC	99.958	5673.203	1157.032	1157.281	1.714	73184	63008	11.917
	CC	99.994	2344.653	486.238	486.243	0.72	69888	65488	15.0
T	FPC	99.958	4884.528	975.945	975.93	1.991	53920	44864	11.0
	CC	99.998	1641.918	358.71	358.706	0.732	50656	47416	14.0
A	FPC	99.979	3948.326	944.737	944.683	2.077	49440	41568	11.0
	CC	99.966	1549.597	334.983	335.26	0.737	47016	44120	14.0
O	FPC	99.948	4264.16	984.048	942.275	2.13	48416	40000	11.0
	CC	99.981	1680.499	325.263	325.152	0.735	45664	452552	14.0
N	FPC	99.988	4022.49	866.384	866.373	2.235	42688	34752	11.0
	CC	99.985	1423.982	284.175	284.275	0.733	40096	37352	14.0
I	FPC	99.954	4294.694	855.847	855.906	2.243	42432	34560	11.0
	CC	99.98	126.274	283.07	283.145	0.742	39384	36880	14.0
R	FPC	99.997	3956.032	825.349	825.355	2.353	39040	31552	10.988
	CC	99.994	1206.931	253.579	253.611	0.723	36288	33976	14.0
S	FPC	99.973	3919.046	812.529	812.432	2.435	37056	29440	10.723
	CC	99.996	1145.379	238.064	238.045	0.713	34488	32224	14.0
H	FPC	99.928	3635.264	780.482	780.726	2.534	34048	27200	10.028
	CC	99.963	1070.522	214.352	214.454	0.696	31872	29768	14.0
D	FPC	99.967	3851.923	688.013	688.0	2.875	27776	20864	10.0
	CC	99.98	764.845	166.913	166.99	0.698	24668	23172	14.0
L	FPC	99.992	3214.502	626.176	626.176	3.195	22496	16384	10.0
	CC	99.993	3214.502	626.176	626.176	3.195	22496	16384	14.0
C	FPC	99.848	2761.325	560.859	561.158	3.511	18720	13632	9.278
	CC	100	538.938	110.968	110.968	0.694	16460	15444	14.0
U	FPC	99.931	2718.624	558.07	558.24	3.609	18176	13088	9.096
	CC	99.986	574.144	108.45	108.458	0.701	16040	14988	14.0
F	FPC	99.995	2603.93	549.101	549.09	3.561	18016	12992	9.085
	CC	99.996	521.575	106.095	106.095	0.688	15940	14976	14.0
M	FPC	99.844	2369.696	500.328	500.861	4.051	14752	10176	9.0
	CC	99.995	344.406	72.338	72.34	0.585	12708	12052	13.803
P	FPC	99.97	2266.496	479.837	479.814	4.18	13664	9216	9.0
	CC	99.978	395.532	68.488	68.524	0.597	11746	11086	13.0
G	FPC	99.866	2263.488	461.661	461.754	4.26	13088	8704	9.0
	CC	99.996	290.569	64.919	64.92	0.599	11130	10556	13.0
W	FPC	99.808	1847.661	413.722	413.888	4.759	10528	6848	8.83
	CC	99.991	228.227	53.818	53.821	0.619	8916	8468	13.0
B	FPC	99.928	1964.704	401.928	402.019	5.05	9920	336	8.329
	CC	99.99	224.234	49.202	49.212	0.618	8186	7742	13.0
Y	FPC	99.924	2187.942	399.16	399.138	5.074	10048	6016	8.264
	CC	99.99	52.762	49.409	49.411	0.628	8084	7614	13.0
V	FPC	99.966	1680.832	324.991	324.979	6.101	6720	3648	8.0
	CC	99.986	154.232	28.35	28.364	0.532	5448	5172	13.0
K	FPC	99.817	1058.579	225.07	225.139	9.013	3552	1536	6.955
	CC	100.0	0.0	0.0	0.0	0.0	2498	2498	12.0
J	FPC	99.983	932.781	161.973	161.977	12.704	2208	576	5.911
	CC	100.0	0.0	0.0	0.0	0.0	1275	1275	11.0
Q	FPC	99.501	786.323	135.952	135.725	14.485	1728	384	5.332
	CC	100.0	0.0	0.0	0.0	0.0	937	937	10.0
X	FPC	99.874	633.178	133.569	133.658	14.28	1568	384	5.321
	CC	100.0	0.0	0.0	0.0	0.0	936	936	10.0
Z	FPC	99.959	766.643	130.664	130.633	15.102	1632	384	5.179
	CC	100.0	0.0	0.0	0.0	0.0	865	865	10.0

TABLE II

Discrete comparison of Fixed Probability Counter with p=1/32 and Floating-Point Counter with d=12