SKETCHES ON SINGLE BOARD COMPUTERS

Ali Osman Berk Şapçı aliosmanberk@sabanciuniv.edu
Computer Science and Engineering, 2015

Egemen Ertuğrul egemenertugrul@yandex.com
Computer Science and Engineering

Hakan Ogan Alpar halpar@sabanciuniv.edu
Computer Science and Engineering, 2016

Kamer Kaya
Computer Science and Engineering

Abstract

Data sketches are probabilistic data structures with mathematically proven error bounds that store information about the datasets using small amount of space using hash functions. They can approximate some predefined questions (i.e. cardinality or frequency estimation) about big data with high accuracy. We chose a data sketch called HyperLogLog and tested it with many different configurations in order to find results that show least errors possible when estimating unique elements in large genomic datasets.

Keywords: Big Data, Data Sketches, Bioinformatics, HyperLogLog, K-Mers

1 Introduction

Estimating the number of unique items in a dataset is called the count-distinct problem. A naïve approach to this would be to keep track of each unique member encountered. However, it becomes a challenge for memory and speed when a big data is processed; it isn't always practical to calculate the exact cardinality. Data sketches or probabilistic data structures are employed to overcome the inefficient use of resources and approximate cardinalities or frequencies with mathematically proven error bounds in a large datasets. These methods are sufficiently accurate when some amount of error is acceptable. The lecture notes of Chakrabarti (2015) presented the basics of the data sketches in a very concise fashion. Some of the most prominent data sketches in the field are HyperLogLog (for cardinality estimation) and Count-Min (for frequency estimation). In this work, we have focused on the count-distinct problem, and therefore HyperLogLog algorithm. This algorithm was proposed by Flajolet et al. (2007) as an extention to LogLog algorithm (Durand and Flajolet, 2003) which was derived from the Flajolet-Martin algorithm (Flajolet and Martin, 1985). HyperLogLog is well-known for its low memory footprint and high accuracy on large number of cardinalities. Heule et al. (2013) presented some improvements for HyperLogLog in order to reduce memory requirements and increase accuracy for some cases of cardinalities.

In the field of Bioinformatics, large genomic datasets are stored through DNA sequencing to be processed later on. Such datasets could reach GBs or even TBs in size. In order to perform

bioinformatics analysis, substrings or subsequences of length k called “k-mers” are read from DNA sequences. It is important to determine the cardinalities or frequencies of certain k-mers in such datasets, however, the computational resources might be limited or insufficient, such as in the single board computers. In this case, data sketches can help immensely to overcome these constraints by estimating with the cost of accuracy.

Zhang et al. (2014) presented the khmer software package for frequency estimation of k-mers using Count-Min sketch and compared the performances of several k-mer counting packages with their proposed model. Rizk et al. (2013) proposed an exact k-mer counting software called DSK (disk streaming of k-mers) which requires a user-defined amount of memory and disk space. We deployed DSK in our work and compared our approximated results with the exact results obtained from DSK. Mohamadi and Birol (2017) proposed ntCard algorithm for estimating k-mer frequencies using ntHash (Mohamadi et al., 2016). During our research, we deployed and analyzed the ntCard algorithm to make speed and accuracy comparisons with the results obtained from our model.

Just like in many other data sketches, a hash function is used in the HyperLogLog algorithm. Since there are many different hash functions used in different conditions, we had to narrow down our options to few non-cryptographic and considerably fast hash functions: Murmur3 Hash, Tabulation Hash, City Hash and Spooky Hash. Instead of picking just one, we decided to compare the results of HyperLogLog algorithm working with these hash functions.

The genomic datasets that we used to conduct our experiments with HyperLogLog were obtained from UCSC Genome Database (Karolchik et al., 2003). We used the four datasets: danRer11, triCas2, apICal1 and droEre2. aplCal (A. californica)1 has size 697Mb, Danrer (D. rerio) as biggest data of our sample has size of 1.6Gb. DroEre2 (D. erecta) is 149 Mb, triCas2 (T. casteneum) is 195Mb. Additionally, we have tested our model with 2 more datasets called est and est1. Data sizes of est and est1 are 97Mb and 89Mb respectively.

For the sake of consistency, we conducted our experiments on a single HPC server with the following specifications: 2 x Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz 16 cores running Ubuntu 16.04.2 LTS.

In the rest of the report, we mention the HyperLogLog algorithm in detail, give the percent error results for each hash function with the corresponding K and bucket bit values, present our outcome from the results, and finally give our conclusion and present potential future work.

2 HyperLogLog on Genomic Data

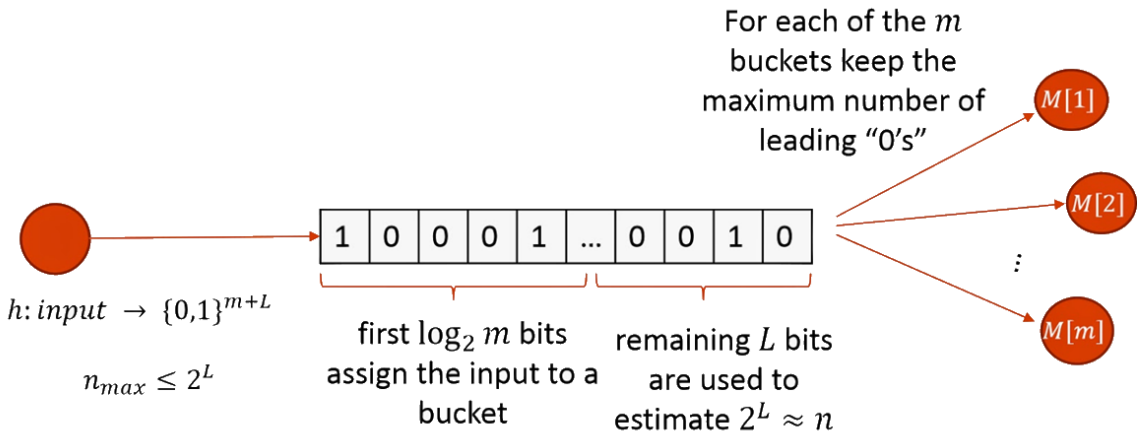


Figure 1: ("How to count distinct?" 2015)

Our implementation of HyperLogLog offers different combinations of hash functions, register sizes (or bucket bits) and k-mers. Program asks the user to choose a hash function between MurMur, Tabulation, Spooky and City Hash, then expects number of bits to construct buckets and k number to estimate cardinality of a particular k-mer size.

HyperLogLog operates as follows: after getting the number of bits (call it p) from user input, construct an array of size 2^p. Later, the dataset whose cardinality will be estimated is read to be hashed. 32 or 64-bit value of a specific element that returned from hash function is used to determine bucket index and probabilistic calculation of cardinality. First binary value m bits of returned hash value are used to determine bucket index (see Figure 1). Number of leading zeros of remained bits will be the value of that bucket index if it is the maximum value that have been seen so far. Every bucket represents a subset of the initial datasets. Values of that buckets gives us a probabilistic intuition to calculate estimation. Each leading zero encountered gives us to a reliable conviction about doubling of number of distinct elements of that subset. Evaluating value of all buckets together gives an accurate estimation.

$$E := \alpha_m \cdot m^2 \cdot \left(\sum_{j=1}^{m} 2^{-M[j]} \right)$$

where

$$\alpha_m := \left(m \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}$$

Figure 2: ("HyperLogLog in Practice: Algorithmic Engineering of a State-of-the-Art Cardinality Estimation Algorithm," 2017)

The formula seen above in Figure 2 gives us the final cardinality estimate(E) is produced from the substream estimates as the normalized bias corrected harmonic mean where m is equal to number of buckets(2^p) M is the array of buckets and α_m is the constant (also known as magic number).
In bioinformatics, researchers need to work with different K-Mer sizes, and as K-Mer size increases calculations become difficult, because of this we started with length 32 and worked our way up to 64. We had 6 different data, some were larger around 1GB and some were lower around 70 mb. We tried to find a correlation between the hash functions and accuracy of the sketch. For this we worked with different parameters, such as length of our data by using different data sizes. Working with different bucket sizes, working with different K-Mers and finally changing the hashes themselves.
Since the files were too large for us to find the cardinality on our own, we used a specific tool that does that. Namely DSK. It is an exact counting tool that's uses the disk to count and find the cardinality. The question becomes then why anyone would use data sketches and not DSK for cardinality counting, and the answer is because DSK can take a long time and requires a lot of space.

We run our experiments by using 32-bit hash functions on k=32, 40, 48, 56 and 64, but 64-bit versions of them also available. Following sections present result of experiments on 32 and 64 k-mer. Other results and corresponding plots can be found in Appendices section.

2.1 Percent Error Results with K-Mer Length 32

By looking at the tables with K-Mer length 32, we can see a lot of fluctuations in the accuracies of hash functions.

ERR	Hash 1 (City)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	28,32903	-0,47752	0,32568	-2,42212	-2,69611	4,360159	13,31973
droEre2	26,08472	9,007605	1,537999	-1,8448	-1,86672	32,91881	10,52685
triCas2	-11,3154	-6,98836	3,641623	1,016213	1,644769	-2,40023	6,423955
est	-11,1559	16,44484	23,94884	22,94607	22,82548	15,00186	14,92213
est1	14,80821	9,404392	13,07921	15,97663	15,39878	13,73344	2,651841
danRer	-23,5561	-4,5972	4,481148	6,296602	5,481442	-2,37883	12,62444

Table 1:32 K-Mer Length City Hash Results

	Hash 2 (Spooky)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	-6,16254	-2,00645	-3,62256	-5,16196	-3,25258	-4,04122	1,635051
droEre2	4,120409	3,588149	-2,51519	-4,48429	-1,55365	-0,16891	3,826184
triCas2	-11,8826	-10,2646	-2,74888	-1,3645	1,601884	-4,93174	5,8509
est	-10,4284	0,804696	12,94201	18,33887	22,61821	8,855077	13,5345
est1	20,01498	26,09756	16,50607	19,32839	16,80599	19,7506	3,864255
danRer	-3,41148	13,75741	14,05929	6,6609	7,035217	7,620267	7,107877

Table 2: 32 K-Mer Length Spooky Hash Results

	Hash 4 (Murmur)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	18,15211	9,009887	1,382242	-3,04857	-2,22886	4,653363	8,924408
droEre2	10,6928	3,861166	-3,92286	-1,46581	-2,67153	1,298752	6,030865
triCas2	5,122158	11,19662	4,075823	0,880328	1,417772	4,53854	4,1233
est	-11,9336	9,145206	22,65226	23,08353	22,12962	13,0154	15,12232
est1	2,439208	11,3283	15,98586	16,92235	15,31637	12,39842	5,961988
danRer	-2,84937	4,091096	11,73827	8,783472	7,06333	5,765358	5,556216

Table 3: 32 K-Mer Length Murmur Hash Results

	Hash 5 (Tabulation)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	-0,81926	-0,31408	-0,95929	-3,14298	-1,57649	-1,36242	1,092333
droEre2	-2,40244	0,668195	-6,52774	-1,18951	-1,8116	-2,25262	2,653063
triCas2	-19,5032	3,875637	0,598706	4,286389	2,253572	-1,69777	10,05973
est	15,23084	7,434267	17,38598	23,71222	22,20439	17,19354	6,456285
est1	18,61013	13,28527	12,86896	15,49735	13,87788	14,82792	2,338436
danRer	-10,8732	1,804279	4,676993	7,592755	7,263135	2,092788	7,613489

Table 4: 32 K-Mer Length Tabulation Hash Results

We can see all hash functions perform better for larger amounts of bits taken. This means that we can reach a more accurate result by setting up an array of 2^10 or 2^14 buckets to reach our optimal accuracy with bits taken. Yet in the other parameters things are not as clear. Looking at the 10 bits column of tables 1, 2, 3, 4 we can see City hash performs the best for "aplCal 1" and Tabulation Hash for triCas2, they also seem to perform the best overall. Yet while all the hashes in this 10 bits column perform around below 10% we have 2 datasets (est and est1) that has almost 20% error rate. This at first led us to think our algorithm was not very accurate because 20% error is too high for our expectations. This led us to test all these data with another, very sophisticated cardinality counting algorithm for K-Mer counting called NTCard, and the results were better than HyperLogLog, yet similar.

ERROR	32	40	48	56	64
triCas2	0.351716	0.38153	0.48135	0.5399386	0.704322
est	14.264171	15.52537	14.83173	16.830817	17.24306
est1	7.0655798	7.579474	8.014691	8.6266012	8.858206
danRer11	0.140161	0.177882			
droEre2	0.2209904	-0.02371	0.104777	0.1375697	0.261346
aplCal1	-0.004707	0.105699	0.098857	0.0807122	0.092264

Table 5: NTCard Results for K-Mer Lengths 32, 40, 48, 56 and 64

By looking at Table 5, we can see even though ntCard performed really well for triCas2 and Danrer11, it failed to perform accurately when it came to "est" and "est1". This proved that the problem was not with our HyperLogLog implementation. Unfortunately, we were not able to obtain DSK results with the danRer11 dataset with K values of 48, 56 and 64, therefore the percent error results do not exist for both ntCard and our model.

2.2 Percent Error Results with K-Mer Length 64

ERR	Hash 1 (City)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	28,32903	9,738833	1,081687	0,17514	-0,87483	4,360159	13,31973
droEre2	-12,6461	-0,30178	0,922904	-0,11187	-1,04152	-13,1783	5,044385
triCas2	16,57566	-2,22379	1,063134	0,069423	1,32826	3,362537	7,517709
est	57,97755	31,57177	23,95072	21,00053	21,82709	31,26553	15,50473
est1	24,54762	8,810022	16,38134	14,90347	15,7816	16,08481	5,615094
danRer							

Table 6: 64 K-Mer Length City Hash Results

	Hash 2 (Spooky)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	10,02768	-1,5066	-4,19433	-3,00216	-2,33141	-0,20136	5,802204
droEre2	6,591349	6,973159	-0,06165	-0,49737	-2,33192	2,134712	4,329402
triCas2	-6,87982	-0,15531	-1,22223	-1,06074	1,72694	-1,51823	3,218392
est	-0,98121	19,00302	24,52954	25,07038	23,91395	18,30713	11,05012
est1	42,87334	33,15801	13,6138	12,93238	15,06248	23,528	13,68579
danRer							

Table 7: 64 K-Mer Length Spooky Hash Results

	Hash 3 (Murmur)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	-0,7054	0,067993	-2,07026	-0,90465	-0,7054	-0,86354	0,770653
droEre2	6,591349	5,772626	-5,32064	-2,24536	-2,02832	0,553931	5,308008
triCas2	44,81134	20,39261	4,395721	1,134941	1,038187	14,35456	18,81048
est	23,28877	16,93833	20,65375	23,33741	23,97426	21,6385	2,920906
est1	6,996839	23,62446	15,80245	16,66608	15,67019	15,752	5,90494
danRer							

Table 8: 64 K-Mer Length Murmur Hash Results

	Hash 4(Tabulation)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	-1,30217	9,532488	-1,5017	-1,32744	-1,70005	0,740226	4,917613
droEre2	-17,1448	-12,1746	-4,26843	-0,41496	-0,8145	-6,96347	7,395477
triCas2	3,139113	6,0111	3,121772	1,547118	1,236432	3,011107	1,892176
est	29,68183	21,32712	24,4255	21,82873	26,39814	24,73226	3,442129
est1	26,64757	4,340913	18,38749	14,22262	14,65305	15,65033	8,052439
danRer							

Table 9: 64 K-Mer Length Tabulation Hash Results

As in the K=32 results, est and est1 results are still not reliable and therefore it is hard to make an outcome. NtCard also gives high erroneous estimations about est and est1 datasets. Except for Spooky Hash, all other hash functions give much more accurate results on K=64 than K=32. This indicates that the conclusion of greater K length leads more accurate results. There is no error higher than %2 when Tabulation Hash is performed and bucket bit number is 14. When the bucket bit is 6, poor results are likely to arise. Improvement of accuracy of Murmur can be seen in Table 8.

3 Conclusion and Future Work

Findings so far suggest that HyperLogLog data sketch for K-mer counting gives more accurate results with bucket bits around 10 and a large distinct element size utilizing Tabulation and City Hashes, yet there are other conditions to consider. Right now, even though our implementation of HyperLogLog saves a lot of space, it runs sequentially, therefore it takes time to process the datasets; parallelization is needed to bring our model to more desirable speeds. Also, in our estimation formula, there is a correction constant that differs for each bucket size which might need some tweaking. Although a file size of 1GB, like the ones we used might seem large at first, file sizes can go much larger in the field of Bioinformatics. Tests need to be done on these types of larger data to have more accurate results and to see which hash function performs the best for HyperLogLog sketch.

References

Chakrabarti, A. (2015). Data stream algorithms. *Computer Science*, *49*, 149.

Flajolet, P., Fusy, É., Gandouet, O., & Meunier, F. (2007, June). Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science* (pp. 137-156). Discrete Mathematics and Theoretical Computer Science.

Durand, M., & Flajolet, P. (2003, September). Loglog counting of large cardinalities. In *European Symposium on Algorithms* (pp. 605-617). Springer, Berlin, Heidelberg.

Flajolet, P., & Martin, G. N. (1985). Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, *31*(2), 182-209.

Heule, S., Nunkesser, M., & Hall, A. (2013, March). HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology* (pp. 683-692). ACM.

Zhang, Q., Pell, J., Canino-Koning, R., Howe, A. C., & Brown, C. T. (2014). These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure. *PloS one*, *9*(7), e101271.

Rizk, G., Lavenier, D., & Chikhi, R. (2013). DSK: k-mer counting with very low memory usage. *Bioinformatics*, *29*(5), 652-653.

Mohamadi, H., Khan, H., & Birol, I. (2017). ntCard: a streaming algorithm for cardinality estimation in genomics data. *Bioinformatics*, *33*(9), 1324-1330.

Mohamadi, H., Chu, J., Vandervalk, B. P., & Birol, I. (2016). ntHash: recursive nucleotide hashing. *Bioinformatics*, *32*(22), 3492-3494.

Karolchik, D., Baertsch, R., Diekhans, M., Furey, T. S., Hinrichs, A., Lu, Y. T., ... & Weber, R. J. (2003). The UCSC genome browser database. *Nucleic acids research*, *31*(1), 51-54.

How to count distinct? (2015, April 29). Retrieved from http://www.itshared.org/2015/04/how-to-count-distinct.html

HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm. (2017, July 27). Retrieved from https://blog.acolyer.org/2016/03/17/hyperloglog-in-practice-algorithmic-engineering-of-a-state-of-the-art-cardinality-estimation-algorithm/

Appendices

Appendix A

Error Results with K-Mer Length 40

ERR	Hash 1 (City)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	28.329031	3.810239	-3.20215	-0.43761	-1.33368	4.360159	13.31973
droEre2	9.83139	6.617879	5.297024	1.503681	-1.16117	22.0888	3.86102
triCas2	-15.96679	3.33533	7.741705	4.51567	2.593122	0.443808	9.382529
est	-6.201545	8.24261	18.21004	24.30403	22.08291	13.32761	12.53216
est1	-14.3671	11.91766	10.22176	13.81715	16.20329	7.558552	12.45723
danRer	4.8578846	12.58754	5.718625	3.807725	4.840399	6.362434	3.545179

	Hash 2 (Spooky)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	7.407527	5.92531	2.9053829	1.013237	-0.70755	3.308781	3.361886
droEre2	-13.563	-6.45241	-3.082123	-2.1849	-0.88737	-5.23396	5.091102
triCas2	5.761526	-6.40233	0.1031843	3.010521	0.732663	0.641113	4.520525
est	8.054934	17.53015	25.255193	26.90899	23.7332	20.29649	7.707679
est1	11.58477	7.487913	14.843526	13.4085	16.96096	12.85713	3.589518
danRer	-10.7519	-4.6079	6.3590817	6.194416	6.499414	0.738615	7.986868

	Hash 4 (Murmur)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	13.86492	-7.66097	-1.17478	-3.30619	-3.654601	-0.38632	8.304327
droEre2	-12.6859	-7.34543	-6.49402	-3.63491	-2.790242	-6.5901	3.902259
triCas2	8.846595	-4.16029	3.705522	3.109671	1.2542494	2.551149	4.690485
est	29.11144	31.49208	22.81277	22.7882	23.077415	25.85638	4.145975
est1	19.26702	7.413903	16.26557	13.61818	14.513049	14.21555	4.371148
danRer	9.112314	10.31699	7.754392	4.966647	5.7694932	7.583967	2.23478

	Hash 5 (Tabulation)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	13.04263	-16.6824	-5.004373	-3.45269	-2.65846	-2.95105	10.59363
droEre2	0.100119	5.314277	-0.661269	-2.23048	-0.11461	0.481606	2.851198
triCas2	19.47696	0.879472	-1.003382	4.196165	2.511616	5.212167	8.203884
est	41.38791	29.35921	19.959875	19.76241	22.1542	26.52472	9.177095
est1	19.26702	7.413903	19.175086	13.98109	13.49226	14.66587	4.897473
danRer	24.07031	1.914857	2.1477857	5.134403	4.502338	7.553939	9.34054

Error Results with K-Mer Length 48

ERR	Hash 1 (City)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	28.32903	6.516474	-5.33696	-4.82313	-3.1649	4.360159	13.31973
droEre2	13.74628	1.535539	-1.37512	-0.75808	-1.59332	11.5553	5.824388
triCas2	-9.34023	-3.95298	3.438084	-1.30559	0.880688	-2.056	4.899762
est	16.95189	18.17088	28.86127	25.40778	22.75572	22.42951	4.961961
est1	31.98969	14.91522	18.19513	16.81133	16.74034	19.73034	6.951525

	Hash 2 (Spooky)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	14.36503	2.447826	-2.65071	-3.94363	-2.804813	1.48274	7.612471
droEre2	20.83286	7.941589	3.756321	2.006307	0.1154852	6.930512	8.292547
triCas2	-12.5275	-9.68022	5.613552	1.042695	1.6761489	-2.77506	7.866835
est	5.504996	24.77914	20.18581	23.03873	22.155735	19.13288	7.795856
est1	24.35485	21.81703	18.66445	14.06185	14.744158	18.72847	4.439919

	Hash 4 (Murmur)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	9.783287	-9.08781	-2.9997	-2.26221	-2.45121	-1.40353	6.866145
droEre2	-19.0768	3.702389	-0.45869	-0.85789	-1.85023	-3.70825	8.849394
triCas2	0.591081	-3.9533	-0.78764	1.096613	1.235561	-0.36354	2.160249
est	28.50798	18.96085	22.20584	22.62528	21.64259	22.78851	3.50152
est1	-4.2162	10.74451	11.74423	14.52561	15.28486	9.616602	7.958847

	Hash 5 (Tabulation)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	15.48912	-13.3996	-4.34403	-0.57503	-0.99549	-0.765	10.44901
droEre2	-12.9192	6.200001	-1.18178	-0.62154	-2.18053	-2.14061	6.873362
triCas2	11.94478	-7.16997	-0.72122	2.096881	0.927842	1.415662	6.889188
est	17.51242	20.24594	22.85868	20.61505	22.26464	20.69935	2.090579
est1	-2.21572	15.22107	16.98623	11.22328	14.98483	11.23994	7.809427

Error Results with K-Mer Length 56

ERR	Hash 1 (City)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	28.32903	8.382565	2.0618745	-1.68298	-1.18425	4.360159	13.31973
droEre2	-6.72477	0.105947	4.7817782	-0.46809	-1.8078	-4.11294	3.693323
triCas2	16.58613	12.81609	3.538529	3.979924	2.655337	7.9152	6.354358
est	43.02778	27.35328	23.862398	24.72415	23.59436	28.51239	8.249247
est1	46.61255	27.46148	15.402295	15.61322	15.8336	24.18463	13.54703

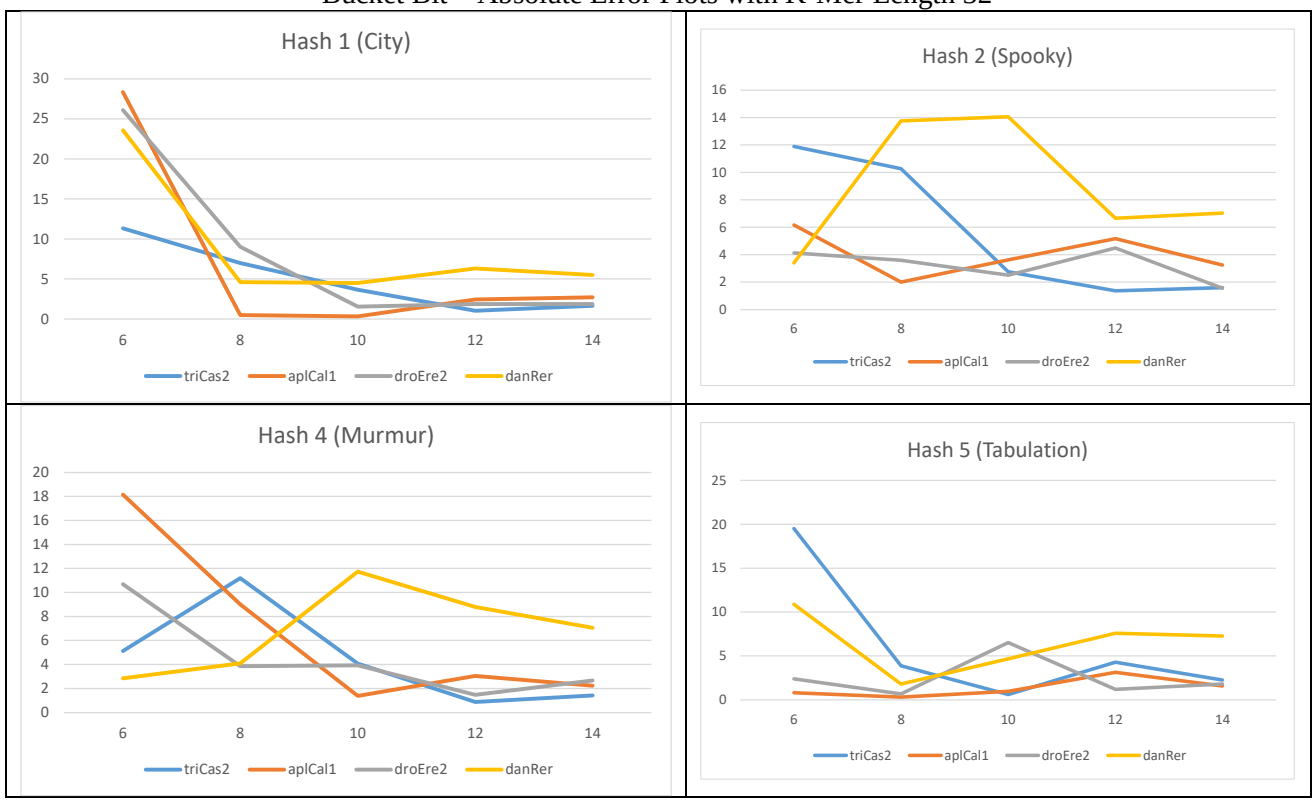
	Hash 2 (Spooky)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	10.44332	1.824752	-1.43371	-4.48699	-0.6393	1.141613	5.667307
droEre2	17.8682	-0.32941	3.163984	-0.07309	-0.38464	4.04901	7.867224
triCas2	28.86708	4.109925	-0.93368	2.519895	0.782926	7.06923	12.3305
est	11.12468	13.06368	22.92657	23.07438	22.50969	18.5398	5.927438
est1	33.0446	19.90242	16.14578	15.39513	15.53324	20.00423	7.519689

	Hash 4 (Murmur)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	4.530359	4.944588	-1.00691	-2.6	-2.81763	0.610082	3.834872
droEre2	11.28798	5.418142	0.779537	-2.31269	-2.71764	2.491067	5.89681
triCas2	9.985381	1.611973	1.148735	2.712825	2.381992	3.568181	3.640041
est	8.630353	9.745224	25.90477	24.36541	24.34799	18.59875	8.623222
est1	14.30599	4.738795	10.76795	15.38187	13.81736	11.80239	4.304218

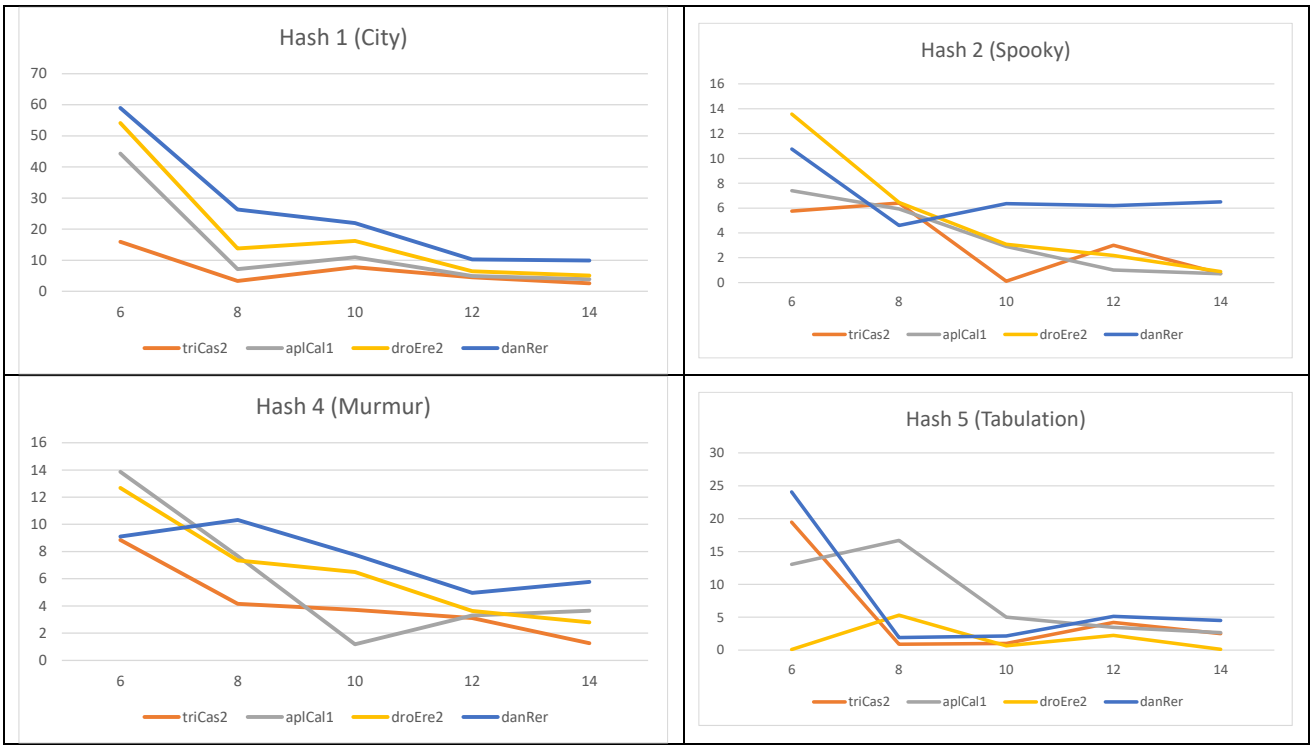
	Hash 5 (Tabulation)						
%	6	8	10	12	14	AVG	STDEV
aplCal1	0.989402	-9.40619	1.814707	-4.603141	-0.80043	-2.40113	4.630091
droEre2	13.13029	-23.7305	0.525524	0.3173075	-2.47667	-2.4468	13.34274
triCas2	38.88303	1.21936	1.343621	2.8215189	1.288516	9.11121	16.65633
est	28.43838	22.6921	24.07895	23.186344	23.70329	24.41981	2.306779
est1	-4.72838	19.05102	14.59442	12.719319	14.39384	11.20605	9.21117

Appendix B

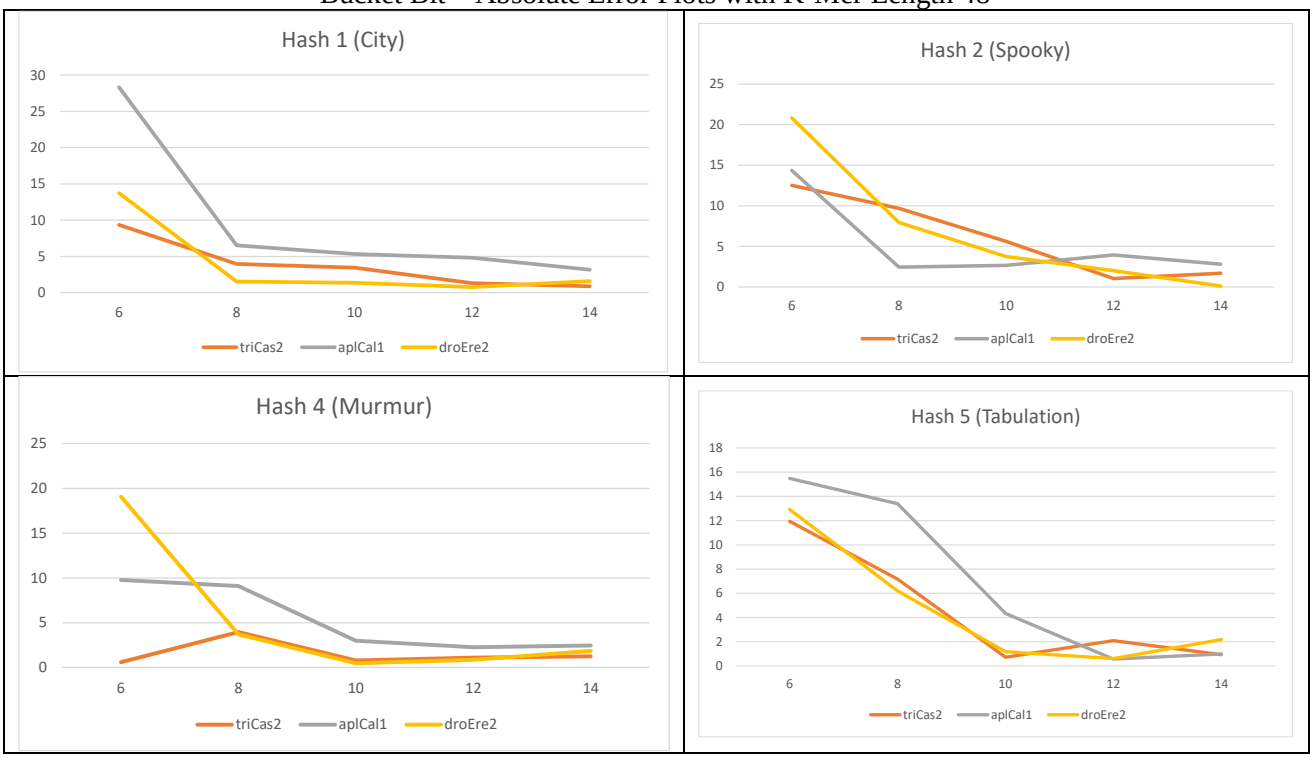
Bucket Bit – Absolute Error Plots with K-Mer Length 32



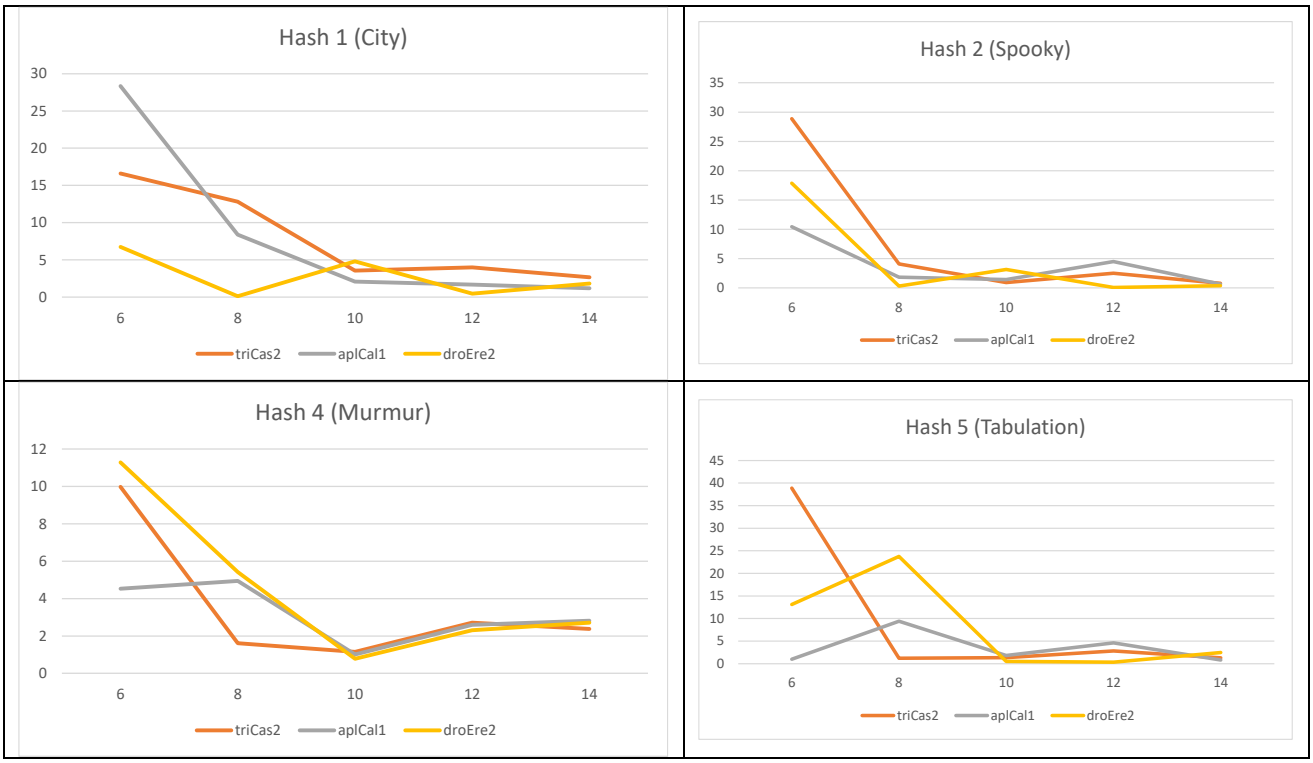
Bucket Bit – Absolute Error Plots with K-Mer Length 40



Bucket Bit – Absolute Error Plots with K-Mer Length 48



Bucket Bit – Absolute Error Plots with K-Mer Length 56



Bucket Bit – Absolute Error Plots with K-Mer Length 64