CO322 **Data Structures and Algorithms**
Lab01 **Hash tables**
E/14/158 Gihan Jayatilaka
03-02-2018

---

1. Objectives

To pick performance metrics for hash functions.

To analyze the performance of existing hash functions.

To propose a new hash function.

2. Performance metrics

The experiment measures "performance" as the inverse of the number of collisions possible for a retrieval of a word form the hash table. The number of collisions is proportional to the number of keys in a particular bucket.

2.1. Mean

Mean is always the same for every hash function for a particular number of buckets and a particular set of words. It is not a useful metric.

2.2. Standard deviation

Standard deviation gives a good idea about how the bucket sizes are distributed. Lower standard deviations implies good performance of a hash function.

2.3. Entropy

Entropy is the randomness of a data set. This is a measure of how well the data is spread into buckets. A uniform distribution gives the highest entropy. Entropy is the best metric for the performance.

2.4. Min-Max

The range between the min and max is an indication of the distribution. Lower the range, better the performance of the hash function. But there can be a case where min or max is an outlier so the range might not give a real insight to the data set.

3. Implementation

- class Driver
- interface HashTable
    - class HashTableImp implements HashTable
        - class HashTableImp1 extends HashTableImp
        - class HashTableImp<2,3,4,5> extends HashTableImp
- class Learn

3.1. Driver

This is the user interface for the program. To run this,

$ java Driver fileName noOfBuckets

3.2 Learn

This is the interface for the adaptive algorithm to generate a hash function suitable for a particular data set. To run this,

$ java Learn fileName noOfLearningIterations noOfBuckets

3.3 HashTableImp

This class contains the basic functionality for all the HashTable implementations.

3.4 HashTableImpX

These classes have the implementation of 5 hash functions.

HashTableImp1	Java's default algorithm
HashTableImp2	Character sum
HashTableImp3	DJB2 algorithm
HashTableImp4	SDBM algorithm
HashTableImp5	Proposed adaptive algorithm

4. Algorithms

Every algorithm cleans the word (removes the non alphabetic characters).

4.1 Java's default algorithm

Input*: word*
Initialize *hash=0*
For *i=1:length_of_word:*
*hash = (hash*31) + character_at_i_in_word*
End for

4.2 Character sum

Input*: word*
Initialize *hash=0*
For *i=1:length_of_word:*
hash = hash + simple_character_at_i_in_the_word
End for

4.3 DJB2 Algorithm

Input: *word*

Initialize *hash=0*

For *i=1:length_of_word:*

  *hash = (hash*33) + character_at_i_in_word*

End for

4.4. SDBM Algorithm

Input: *word*

Initialize *hash=0*

For *i=1:length_of_word:*

  *hash = (hash*65599) + character_at_i_in_word*

End for

4.5. Proposed adaptive algorithm

This algorithm uses a matrix of parameters

  $\text{parameters}_{10x26}$

This algorithm has two parts.

  1. Adapting the hash function parameters for a given text
  2. Hashing

4.5.1. Adapting step

Initialize *$parameters_{10x26}$ as $zeros_{10x26}$,buckets*

Inputs *set_of_words,number_of_buckets,number_of_learning_iterations*

For *i=1:number_of_learning_iterations*

  pick random word from set_of_words

  calculate hash_code for word

  choose the bucket_to_put_word

  if *bucket_to_put_word has the lowest number of words*

  then *put the word to bucket_to_put_word*

  else

    choose the bucket_with_least_words

    choose a_random_index < length of word

    set parmeters[a_random_index][character_of_word_at_that_index] in a way that the word will go to the bucket_with_least_words

    rehash everything in buckets

  end if

End for

4.5.2. Hashing step

Input *word*

Initialize *hash=0*

Data *$parameters_{10x26}$*

For *i=1:minimum of length_of_word and 10*

hash = hash + parameter[i][character_at_i]
End for

5. Performance

The classical algorithms are working in the same way on both Text_1 and Text_2. The adaptive algorithm learns on Text_1 and works on Text_1 and Text_2.

5.1. No of buckets = 10

On Text1

Algo	Mean	StdDev	Entropy	Min-Max
1.	449.1	166.149	2.232855	195-724
2.	449.1	146.11943	2.248083	191-724
3.	449.1	240.03767	2.1691098	184-907
4.	449.1	188.55739	2.21459	197-828
5.(iter=50)	449.1	215.6745	2.1695209	120-736
5.(iter=250)	449.1	164.6842	2.236907	196-772
5.(iter=500)	**449.1**	**144.63158**	**2.2520227**	**262-772**
5.(iter=1000)	449.1	141.33113	2.2518666	220-657
5.(iter=1500)	449.1	175.48076	2.2341187	226-896
5.(iter=2500)	449.1	159.2774	2.2413511	192-820

On Text2

Algo	Mean	StdDev	Entropy	Min-Max
1.	246.9	95.52638	2.2355254	158-447
2.	246.9	80.49528	2.252941	146-430
3.	246.9	89.73789	2.2396705	142-422
4.	246.9	92.9467	2.2417564	151-496
5.(iter=50)	246.9	168.68045	2.096052	64-674
5.(iter=250)	246.9	75.57572	2.2516952	103-345
5.(iter=500)	246.9	83.036674	2.2464123	147-388
5.(iter=1000)	246.9	77.36983	2.252501	145-343
5.(iter=1500)	**246.9**	**60.665394**	**2.2728636**	**162-356**
5.(iter=2500)	246.9	71.18069	2.2579696	129-340

5.2. No of buckets = 32

On Text1

Algo	Mean	StdDev	Entropy	Min-Max
1.	140.34375	128.05533	3.165325	37-582
2.	140.34375	94.9018	3.2752783	43-465
3.	140.34375	128.41795	3.170238	42-674
4.	140.34375	128.05533	3.165325	37-582
5.(iter=50)	140.34375	127.0816	3.1334953	23-530
5.(iter=250)	140.34375	100.91602	3.2592146	48-510

5.(iter=500)	140.34375	97.341736	3.2769637	43-524
5.(iter=1000)	**140.34375**	**92.94239**	**3.29794**	**42-504**
5.(iter=1500)	140.34375	107.36696	3.2533555	44-542
5.(iter=2000)	140.34375	117.033035	3.2202477	45-557

On Text2

Algo	Mean	StdDev	Entropy	Min-Max
1.	77.15625	42.563263	3.3314443	30-182
2.	77.15625	39.911697	3.3420208	27-172
3.	77.15625	48.721207	3.3013346	15-272
4.	77.15625	42.563263	3.3314443	30-182
5.(iter=50)	77.15625	95.46864	2.8952663	4-392
5.(iter=250)	77.15625	43.208866	3.3206692	22-186
5.(iter=500)	77.15625	41.99635	3.338179	30-210
5.(iter=1000)	**77.15625**	**38.39767**	**3.3512952**	**30-178**
5.(iter=1500)	77.15625	38.679855	3.3463337	17-191
5.(iter=2000)	77.15625	40.65257	3.3426716	31-177

<u>5.3. No of buckets = 33</u>

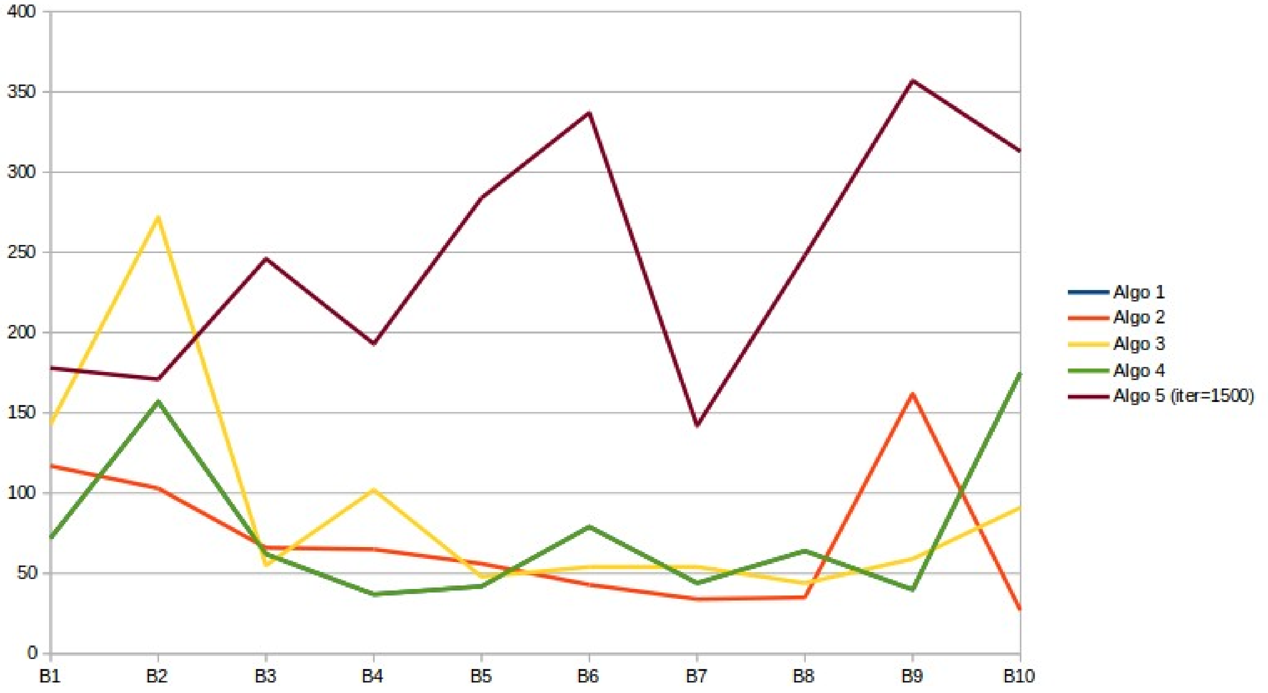
On Text1

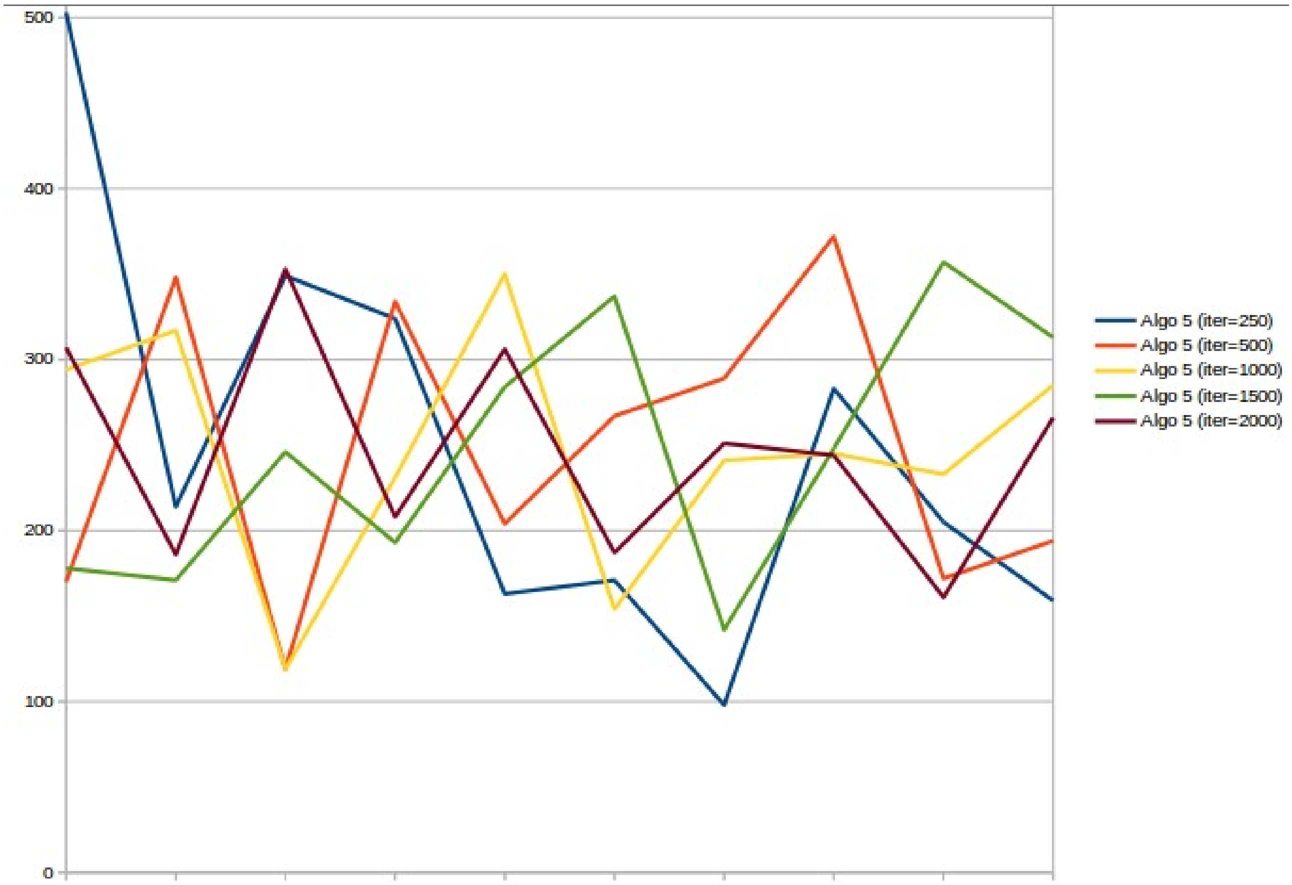
Algo	Mean	StdDev	Entropy	Min-Max
1.	136.09091	104.02778	3.271269	39-469
2.	136.09091	97.82742	3.2880638	38-490
3.	136.09091	223.73056	2.4428327	0-1070
4.	136.09091	99.21949	3.2857006	31-493
5.(iter=50)	136.09091	133.41945	3.0825748	7-502
5.(iter=250)	136.09091	117.21189	3.2471395	33-689
5.(iter=500)	136.09091	97.43634	3.2858996	37-465
5.(iter=1000)	136.09091	103.10641	3.2788007	38-535
5.(iter=1500)	**136.09091**	**93.729904**	**3.3197024**	**53-536**
5.(iter=2000)	136.09091	99.44432	3.2899065	37-529

On Text2

Algo	Mean	StdDev	Entropy	Min-Max
1.	74.818184	44.196842	3.3524072	30-229
2.	74.818184	43.811203	3.3449507	29-206
3.	74.818184	115.48092	2.5600593	0-533
4.	74.818184	49.92446	3.3150337	32-242
5.(iter=50)	74.818184	89.47574	3.0062394	5-480
5.(iter=250)	74.818184	41.785496	3.3547575	29-200
5.(iter=500)	74.818184	47.91752	3.3310473	24-247
5.(iter=1000)	**74.818184**	**37.038437**	**3.3732417**	**21-160**
5.(iter=1500)	74.818184	42.28579	3.3476741	22-194
5.(iter=2000)	74.818184	38.43803	3.3695135	13-171

5.4 Bucket fill graphs for 10 Buckets





6. Conclusions

- In every case, the proposed algorithm outperforms all the traditional algorithms at least at a particular number of learning iterations.
- The proposed algorithm's performance peaks at a particular number of learning iterations and start dropping.
- For lower number of learning iterations, the proposed algorithm performs better on the Text1 which it learn on.
- For higher number of learning iterations, the proposed algorithm performs better on the Text2 than the Text1 it learns from. (UNEXPLAINABLE!)
-