Assignment 3 - Most Frequent and Less Frequent Words

108287 - Miguel Belchior Figueiredo

***Abstract* – This report presents a comparative analysis of three algorithms to identify most/least frequent words in text files. The analysis includes both an exact and an approximate counting algorithm, as well as an algorithm for detecting frequent items in data streams, the Lossy-Count Algorithm. All algorithms are examined through design and an experimental analysis of the computational efficiency and limitations of the algorithms is performed regarding some points, such as: whether the same most and least frequent words are identified and ranked in the same relative order, memory consumption and execution time.**

I. Introduction

The frequent item problem, also known as the "heavy-hitters" problem, involves, given a sequence of items, identifying those which occur most frequently. In this assignment, the task of identifying frequent words in book text files — viewed as a sequence or stream of words — is a direct application of this "heavy-hitters" problem and is accomplished by implementing three different approaches:

- **Exact counter** which accurately computes the exact number of occurrences of each item (word) in a dataset or stream (book text file);
- **Approximate counter** - Probabilistic Algorithm that allows counting a large number of events using a very small amount of memory by providing an estimated count. The specific approach conducted in this assignment is an **approximate counter with fixed probability $\frac{1}{2}$**, meaning that for each event/item there is a probability of $\frac{1}{2}$ of incrementing the respective counter;
- **Data Stream Algorithm** - Algorithm designed to handle data arriving in a streaming fashion processing each element on the fly as it appears. The goal of these algorithms is to compute some function $\Phi(\sigma)$ over an input stream σ efficiently, utilizing sublinear resources in both space and time relative to the size of the input and making just one pass over the input stream. The data stream algorithm conducted in this assignment is the **Lossy-Count algorithm**. The Lossy-Count algorithm is an efficient algorithm for approximating the frequency of items in data streams with limited memory [1]. In the context of this assignment, the stream of items would correspond to the sequence of words in the book. The reason why the algorithm is called "Lossy" is due to one of its properties of periodically discarding buckets based on some delta Δ value, thus approximating the counts of frequent items with some tolerance for error.

II. Methodology

In order to obtain data for the computational experiments and to simulate data streams, text files from the same book in different languages were obtained from Project Gutenberg [2]. The following versions of the book *Romeo and Juliet*, by William Shakespeare, were used:

- Romeo and Juliet by William Shakespeare, in English [3];
- Romeo und Julia by William Shakespeare, in German [4];
- Roméo et Juliette by William Shakespeare, in French [5].

All of the previously mentioned books followed the same preprocessing steps prior to the computational experiments:

- **Removal of Project Gutenberg file header and tail** — Each Project Gutenberg file begins and ends with a standard note with metadata about the file, licensing information, among other details, all of which were removed to focus solely on processing the actual text of each book;
- **Removal of stop-words and punctuation marks** — Puncutation marks were identified using python's `string` module [6], while the list of stop words for each language was obtained from python's NLTK package [7];
- **Lowercase Normalization** — Convert all letters to lowercase to ensure case variations do not impact the count.

III. Algorithms Design

A. Exact Counter Algorithm

As the name suggests, the Exact Counter Algorithm computes the exact counts of each word in the book. The algorithm takes as input the list of words resulting from reading the book text file and returns a map that associates each word with its corresponding count. The algorithm's behavior can be expressed through the following pseudocode:

Algorithm 1 Exact Counter

```
1: Input: words_list
2: Output: word_count_map

3: word_count_map = empty map

4: for each word in words_list do
5:     if word ∉ keys(word_count_map) then
6:         word_count_map[word] = 1
7:     else
8:         word_count_map[word] += 1

9: return word_count_map
```

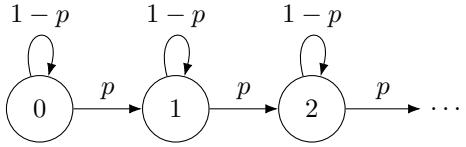
Firstly, the algorithm initializes an empty map `word_count_map`. Then, it iterates through all words in the list. If the current word is not already a key of the `word_count_map` (i.e. the word is encountered for the first time), then initialize its value count to 1. Otherwise, increment the count. After parsing through all words in the list, return the `word_count_map`.

Although this is a relatively simple algorithm, it is utterly necessary for this assignment, serving as a baseline for most of the computational experiments and providing the optimal and exact word count.

B. Fixed Probability Counter Algorithm

Although the exact counter algorithm ensures exact word counts within a given document, it does so at the expense of higher space requirements. The Fixed Probability Counter algorithm addresses this limitation and offers a more efficient solution that allows to count a large number of words with a smaller amount of memory:

1. It produces an estimated count by incrementing the counter of each word with a probability `p`. Therefore, the state diagram bellow represents the evolution of the counter value for each word for a probability `p`;



2. The final word counts must be scaled by the inverse of the probability to obtain the true final estimate.

Compared to the previous algorithm, this algorithm takes an additional parameter `probability` as input which indicates the probability to increment the counter every time a word appears. To achieve this, on line 5, it generates a random float between 0 and 1 (pseudo-random), incrementing the counter if the generated value is less than the specified `probability`. The rest of the algorithm, including iterating over each word and the steps taken to increment the counters, are similar to the last algorithm. The algorithm's behavior can be expressed through the following pseudocode:

Algorithm 2 Fixed Probability Counter

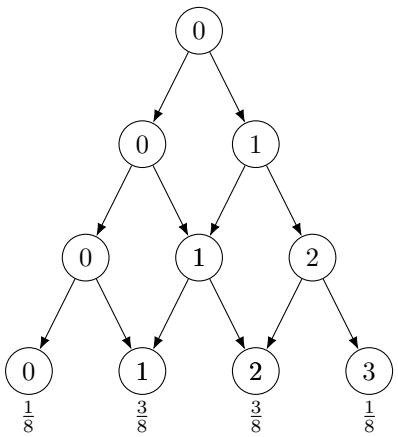
```
1: Input: words_list, probability
2: Output: word_count_map

3: word_count_map = empty map

4: for each word in words_list do
5:     if randomFloat[0, 1] < probability then
6:         if word ∉ keys(word_count_map) then
7:             word_count_map[word] = 1
8:         else
9:             word_count_map[word] += 1

10: return word_count_map
```

For this assignment, an approximate counter with fixed probability $\frac{1}{2}$ was requested to be implemented, which theoretically results in the following binary tree that illustrates the evolution of the counter value for a word in terms of probabilities, given $p = \frac{1}{2}$:



C. Lossy-Count Algorithm

Similarly to the Fixed Probability Counter, the Lossy-Count Algorithm also solves the "heavy-hitters" problem by providing an approximation of the real value count for each of the words in the document. However, the Lossy-Count Algorithm, being specifically designed to identify frequently occurring items in data streams, has as one of its main concerns/advantages memory efficiency. In addition, Lossy Counting provides a guaranteed approximation of the frequencies with an overestimation (no underestimates) by no more than $\epsilon \cdot n$, where ϵ and n correspond to the error parameter and the number of processed elements of the data stream, respectively. This guarantee simply doesn't exist on the last algorithm, where the accuracy of the approximation depends not only of the elements in the stream but also on the random component of the algorithm.

Differently from the last algorithms, the implementation of the Lossy-Count Algorithm followed a class-based approach, which is illustrated in the pseudocode marked as Algorithm 3. This pseudocode (as well as the implemented code) were based on the work of Cormode and Hadjieleftheriou [8].

As in any class in object oriented programming, the constructor initializes the classes parameters:

- **k** — Inverse of the error parameter ϵ. The error parameter epsilon determines the maximum allowable error (in an overestimate) of the frequency count as $\epsilon \cdot n = n/k$, where n is the current number of processed elements. Therefore, a smaller value of ϵ (and consequently an higher value of k) leads to more accurate frequency counts, but requires more memory;
- **delta** — Δ corresponds to the maximum possible error in the frequency of the element and is the result of the application of the formula $\lfloor n/k \rfloor$, where n is the current number of processed elements. As the formula suggests, this value evolves as new elements of the stream are processed;
- **buckets** — The buckets or bins that store the words and their counts;
- **n** — The total number of processed items, which is increased every time an item from the stream (in this case a word) is processed. After parsing through all words in the list, return the word count map.

Whenever a new item appears in the stream, the `processItem` function is called, which can be divided in the following steps:

1. Firstly, on line 7, the number of processed items `this.n` is incremented;
2. Then, if the current item being processed is already a key of the buckets map (i.e. it's not the first time the item/word appears after the last cleanup) increment the count. Otherwise, initialize its value count to $1 + \Delta$;
3. Afterwards, if a new value of delta is computed through the application of the previously mentioned formula $\lfloor n/k \rfloor$, update the current delta and perform a cleanup. This cleanup, also called in some references as pruning, iterates through and removes all bucket items whose frequency is smaller than Δ. Furthermore, bear in mind that the following formula `this.n` mod `this.k` == 0, that verifies if `n` is a multiple of `k` is logically equivalent and could be interchanged with the boolean expression used in line 13.

The function `getFrequentItems` returns the elements in the buckets, and therefore in the data stream, whose frequency exceeds a user-given threshold that defines what is considered frequent. For example, if the goal was to solve the majority problem, the specified threshold would be half of

Algorithm 3 LossyCounting

```
1:  function CONSTRUCTOR(k)
2:      this.k = k
3:      this.delta = 0
4:      this.buckets = empty set
5:      this.n = 0

6:  function PROCESSITEM(item)
7:      this.n += 1

8:      if item ∈ keys(this.buckets) then
9:          this.buckets[item] += 1
10:     else
11:         this.buckets[item] = 1 + this.delta

12:     candidate_delta = ⌊this.n / this.k⌋
13:     if candidate_delta ≠ this.delta then
14:         this.delta = candidate_delta
15:         for each item in keys(this.buckets) do
16:             if this.buckets[item]<this.delta then
17:                 delete this.buckets[item]

18: function GETFREQUENTITEMS(threshold)
19:     frequent_items = empty map
20:     for item, count in Items(this.buckets) do
21:         if count ≥ threshold then
22:             frequent_items[item] = count
23:     return frequent_items

24: function GETNMOSTFREQUENTITEMS(n)
25:     sorted_items = SORTBYCOUNTDESCENDING(Items(this.buckets))
26:     return GETTOPN(sorted_items)
```

the previously mentioned `n` parameter.

Lastly, the function `getNMostFrequentItems` returns the n most frequent elements in the buckets, sorted by their respective count.

IV. Experimental Analysis

A. Exact Counter Algorithm

In order to analyze the computational efficiency and limitations of the developed approaches a series of computational experiments was performed. The experiments were carried out for the book Romeo and Juliet, by William Shakespeare, in three different languages: English, German and French.

A.1 Word Frequency Analysis

The Exact Counter Algorithm was used as the baseline for the number of occurrences of each word for every book. The results for this algorithm correspond to the exact amount of times a given word appears in the words list retrieved after the pre-processing step (removal of Project Gutenberg file header and tail, removal of stopwords and punctuation marks and lowercase normalization) and were validated/asserted with the return value of python's `Counter` object [9]. Table I illustrates the exact count value of the first 20 words — in order of first appearance — for each of the books in the three different languages.

TABLE I
Word Count Metrics — Exact Counter Algorithm
First 20 words

Romeo and Juliet English		**Roméo et Juliett** French		**Romeo und Julia** German	
Word	Count	Word	Count	Word	Count
tragedy	1	roméo	140	romeo	254
romeo	298	juliette	73	julia	139
juliet	178	personnages	3	william	2
william	1	escalus	1	shakespeare	3
shakespeare	1	prince	25	übersetzt	2
contents	1	vérone	14	august	2
prologue	3	paris	5	wilhelm	2
act	13	jeune	39	schlegel	2
scene	51	seigneur	44	personen	2
public	5	parent	9	escalus	1
place	16	montaigu	39	prinz	20
ii	12	capulet	65	verona	9
street	10	chefs	1	graf	15
iii	12	deux	58	paris	37
room	8	maisons	7	junger	4
capulet's	22	ennemies	1	edelmann	3
house	32	vieillard	2	verwandter	4
iv	10	oncle	5	prinzen	5
v	10	fils	22	montague	34
hall	8	mercutio	28	capulet	128

Across all different books the first words refer to the protagonists of the book, which are included in the respective titles: "The Tragedy of Romeo and Juliet", "Roméo et Juliette" and "Romeo und Julia" — as one can conclude only the english title as the word "tragedy" with count 1. Afterwards, the author of the book William Shakespeare is referenced on both the english and german versions of the book. Another clear observation that we can conclude from Table I is that the english book contrary to the french and german counterparts does not start with the personage presentation, but instead with the contents of the book: "prologue", "act", "scene", etc... — the personage presentation is made after that and can only be seen in the full table included in the project result's files delivered with this report. In the personage presentation of the french and german books there is a very high number of matching words with similar counts: "personnages" and "personen", "escalus" (name of a character), "prince" and "prinz", "vérone" and "verona", etc. There are also common words such as capulet (for House of Capulet) that differ a bit on their counts probability due to linguistic differences or differences in the notes of the translator/edition. Speaking of translation, on the german version there is a clear reference to the translator in the words: "ubersetzt", "august", "wilhelm" and "schlege" — as in translated by August Wilhelm von Schlegel, which appears in the beginning and end of the book.

A more insightful approach would be to observe the top 20 and bottom 20 words along with their respective counts for each language, which is displayed in Tables II and III, respectively.

TABLE II
Word Count Metrics — Exact Counter Algorithm
Top 20 words (sorted)

Romeo and Juliet English		**Roméo et Juliett** French		**Romeo und Julia** German	
Word	Count	Word	Count	Word	Count
romeo	298	a	177	romeo	254
thou	277	si	159	julia	139
juliet	178	plus	154	capulet	128
thy	170	roméo	140	wärterin	103
nurse	146	bien	129	o	102
capulet	141	cette	123	mercutio	77
love	136	mort	106	benvolio	68
thee	135	cest	103	nacht	68
shall	110	frère	97	gräfin	66
lady	109	comme	96	ja	60
friar	104	tout	89	ab	60
come	94	nuit	76	wohl	58
mercutio	83	note	75	kommt	58
good	80	juliette	73	tybalt	55
benvolio	79	où	73	liebe	53
enter	75	dun	72	herr	51
go	75	signora	67	schon	48
i'll	71	encore	66	lorenzo	47
tybalt	69	capulet	65	tod	42
death	69	faire	64	paris	37

Again at Table II Romeo and Juliet are among the most frequent words across all books, "romeo" being the top-performer in the english and german counts — 298 and 254, respectively. The english version of the book with words such as "thy" (170) and "shall" (110) reflect Shakespearean style, which are not present in the respective translations. One can also conclude a closer count and word matching on english and german versions with words such as "romeo" (298-254), "juliet"/"julia" (178-139), "capulet" (141-128), "nurse"/"wärterin" (146-103). There are also other words, which although have the same meaning, their counts vary greatly: "death"

(69), "mort"(106) and "tod" (42), "lady"(109) and "signora"(67), "bien" (129) and "wohl" (58). Therefore, although it is the same book, which results in multiple common words with some similar frequencies, linguistic differences and variations in translator or edition styles result in many different words within the top k word counts across the various versions.

As previously mentioned Table III presents the bottom 20 word counts, which essentially displays words than were only registered once — only some of them, as these single-occurrence words account for roughly 50% of the total list of words (rough estimate by observation of the final output file). It's important to note that the order of the words in this sorted format preserves the initial order when the count is the same (in this case 1). Thus, even at this table, it's possible to find similarities between the different languages in the following word sets, such as "sacrifices" and "opfer" or "demand" and "fordern". At the french book, we can also see another reference to project gutenberg (not included in project gutenberg tail but instead added by the french and german translators — therefore not removed by parser) and to the book author. There are also some unusual words on these tables, which makes sense since they only appear once, such as "'pothecary" and "princelaurore".

TABLE III
WORD COUNT METRICS — EXACT COUNTER ALGORITHM
BOTTOM 20 WORDS (SORTED)

Romeo and Juliet English		**Roméo et Juliett** French		**Romeo und Julia** German	
Word	Count	Word	Count	Word	Count
departed	1	montaiguvoyez	1	nachgesehn	1
county's	1	verge	1	leibgedinge	1
rais'd	1	détruire	1	fordern	1
friar's	1	douaire	1	vermag	1
writes	1	montaiguet	1	klarem	1
'pothecary	1	élever	1	bildnis	1
therewithal	1	pur	1	fertgen	1
scourge	1	connue	1	solang	1
winking	1	napprochera	1	bilde	1
brace	1	capuletroméo	1	liebevollen	1
punish'd	1	chétives	1	opfer	1
jointure	1	expiations	1	zwistigkeiten	1
demand	1	inimitiés	1	verhüllt	1
statue	1	princelaurore	1	offenbart	1
whiles	1	aventures	1	ferner	1
figure	1	end	1	erteilen	1
sacrifices	1	project	1	verdarben	1
glooming	1	gutenbergs	1	liebende	1
pardon'd	1	william	1	projekt	1
punished	1	shakespeare	1	etextes	1

A.2 Memory Usage Analysis

Additionally, the memory usage of this algorithm as each word is processed has been tracked by using `asizeof` function of the Pympler library [10]. This function returns the combined (approximate) size in bytes of the `word_count_map` used by the algorithm. It is important to note that the returned size includes a recursive calculation of the sizes of all objects referred to by the dictionary, encompassing both the keys and value sizes, providing a more accurate measurement of the memory footprint of the data structure. Table IV illustrates the progression of memory usage as each word is processed, for the english book. Figure 1 provides a more general overview, showing the memory usage in bytes plotted against the number of processed words. For clarity, only a subset of the original lines of the corresponding table are shown, as the full set of lines could not be accommodated for the size of this document.

TABLE IV
MEMORY USAGE BY WORD STATISTICS
ROMEO AND JULIET ENGLISH

Current Word	Current Word Count	Memory Usage	# Distinct Words
		64	
tragedy	1	320	1
romeo	1	376	2
juliet	1	432	3
william	1	488	4
shakespeare	1	552	5
contents	1	744	6
prologue	1	808	7
act	1	864	8
scene	1	920	9
public	1	976	10
place	1	1312	11
scene	2	1344	11
ii	1	1400	12
street	1	1456	13
scene	3	1456	13
iii	1	1512	14
room	1	1568	15
capulet's	1	1664	16
house	1	1720	17
...	...	...	...
scene	6	1976	21
open	1	2568	22
place	2	2568	22
...	...	...	...
story	2	383016	3734
woe	13	383016	3734
juliet	178	383016	3734
romeo	298	383016	3734
exeunt	34	383016	3734

The following conclusions can be drawn from the analysis of Table IV:

1. Initially, the empty `world_count_dictionary` with no words (keys) or counts (values) occupies 64 bytes;
2. When a python dictionary is created it has an initial capacity. As the dictionary grows and fills up, Python checks if the load factor has reached a threshold — 2/3 for python dictionaries [11]. That being the case Python allocates a larger contiguous block of memory and rehashes the existing keys to new positions, therefore resizing the hash table. Although, Pympler's `asizeof` calculates the recursive total size of the object this can still be seen in sudden spikes in memory consumption that don't result only from the size occupied by the key and value. Examples of evident situations where this occurs are as follows:
   (a) Insertion of word "tragedy" increasing from 64 bytes to 320 bytes, resulting in a 256 byte difference;
   (b) Insertion of word "place" increasing from 976 bytes to 1312 bytes, resulting in a 336 byte difference;
   (c) Insertion of word "open" increasing from

1976 bytes to 2568 bytes, resulting in a 592 byte difference.

As expected these new allocations of memory only occur after a new word is found (`current word count` is 1 and `# Distinct Words` increases in Table IV), with each subsequent allocation increasing in size.

3. As Pympler's `asizeof` calculates the recursive total size of the object it's easy to verify that not all words occupy the same space. For instance, a relatively small sized word such as "romeo", "juliet" and "william" occupies roughly 56 bytes — this conclusion, as well as the next ones, can be achieved by checking the memory usage jumps from 320 to 376 to 432 to 488 on Table IV and by calling the `asizeof` directly on these words. However, a slightly larger word such as "shakespeare" occupies 64 bytes of memory.
4. Lastly, another important thing to note is that incrementing the word count of an initialized dictionary entry generally does not increase the memory usage as the increment in the integer variable is not sufficient to require additional memory allocation. This can be verified in the last 8 lines of Table IV as the memory consumption didn't increase after incrementing the counters for already existing words.

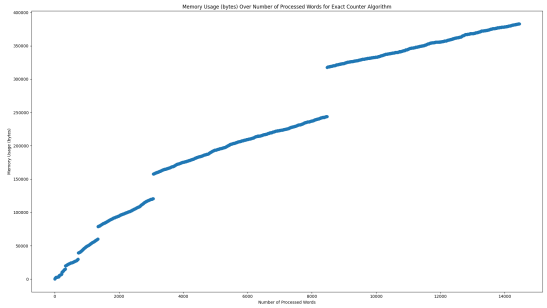


Fig. 1 - Memory Usage (B) by Number of Processed Words for Exact Counter Algorithm — Romeo and Juliet English

In Figure 1 it's clear that the memory usage increases linearly with the number of words processed, as new entries (new words) are constantly being added to the dictionary. It's also evident that the memory consumption spikes (breaks in continuity) shown in the chart are caused by the rehashing/resizing operations mentioned earlier (upon reaching the load factor limit), where new memory blocks are allocated.

In Figures 2 and 3 are plotted the memory usage by the number of processed words of the Exact Counter Algorithm for the French and German books, respectively. The plotted graphs exhibit the same pattern observed on Figure 1. Both the French and German versions take up more space, as evident from the scale on the y-axis, occupying over 400000 bytes and 700000 bytes, respectively. This is due to the fact that the French and German languages are more accentuated than the English one, occupying more space due to their encoding — for instance, "cafe" occupies 56 bytes, while "café" occupies 80 bytes. Furthermore, the higher memory consumption for the French book comparatively to the other two books is also due to the overall higher number of processed words just under 20000 words.

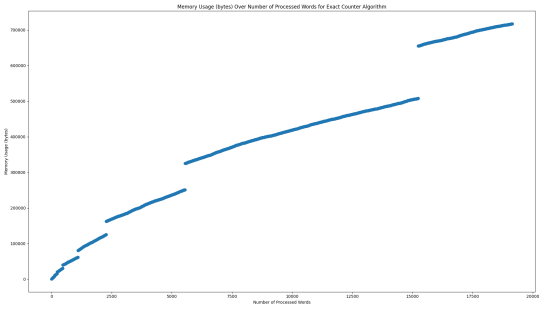


Fig. 2 - Memory Usage (B) by Number of Processed Words for Exact Counter Algorithm — Roméo et Juliet French

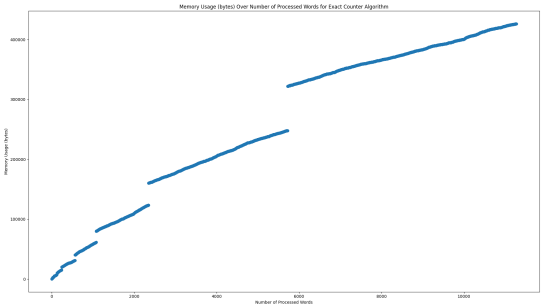


Fig. 3 - Memory Usage (B) by Number of Processed Words for Exact Counter Algorithm — Romeo und Julia German

Table V illustrates the execution time for each of the books relatively to the number of processed words. As expected, the execution time increases with the number of processed words. Thus, the Exact Counter algorithm performs the fastest for the german book.

TABLE V
EXECUTION TIME FOR EACH OF THE BOOKS
EXACT COUNTER ALGORITHM

Metric	German	English	French
Execution Time (s)	0.00074	0.00104	0.00134
Number of Processed Words	11273	14456	19149

B. Fixed Probability Counter Algorithm

B.1 Word Frequency Analysis - 10,000 Iterations

In order to compare the performance/accuracy of the Fixed Probability Counter Algorithm with $p = \frac{1}{2}$ with the Exact Counter Algorithm the following metrics were computed on Table VI over the course of 10000 iterations:

1. **#Occurences** — Number of times the word is counted at least once across all 10,000 iterations;
2. **Mean Count** — The mean count value of the Fixed Probability Algorithm over 10,000 iterations. This is the value contained by the `word_count_map` dictionary prior to multiplying by the inverse of probability (final estimate);
3. **Mean Estimated Count (MEC)** — The estimate count based on the value of mean count, according to the following formula:

$$MEC = MeanCount \cdot \frac{1}{\frac{1}{2}} = MeanCount \cdot 2 \quad (1)$$

4. **Expected Count** — The exact count from the Exact Counter Algorithm corresponds to the expected value;
5. **MEC Absolute Error** — The absolute error based on the MEC and the Expected Count, according to the following formula:

$$MECAbsoluteError = |MEC - ExpectedCount| \quad (2)$$

6. **MEC Relative Error** — The relative error based on the MEC and the Expected Count, according to the following formula:

$$MECRelativeError = 100 \cdot \frac{|MEC - ExpectedCount|}{ExpectedCount} \quad (3)$$

7. **Mean Absolute Error** — The mean of the absolute errors of all 10000 iterations for the given word, according to the formula:

$$MeanAbsoluteError = \frac{1}{n}\sum_{i=1}^{n} |c_i - ExpectedCount| \quad (4)$$

8. **Max Absolute Error** — The maximum absolute error for a specific word over 10,000 iterations;
9. **Min Absolute Error** — The minimum absolute error for a specific word over 10,000 iterations;
10. **Mean Relative Error** — The mean of the relative errors of all 10000 iterations for the given word, according to the formula:

$$MeanRelativeError = \frac{1}{n}\sum_{i=1}^{n} \frac{|c_i - ExpectedCount|}{ExpectedCount} \quad (5)$$

11. **Max Relative Error** — The maximum relative error for a specific word over 10,000 iterations;
12. **Min Relative Error** — The minimum relative error for a specific word over 10,000 iterations.

TABLE VI
PERFORMANCE METRICS OF FIXED PROBABILITY COUNTER ALGORITHM OVER 10,000 ITERATIONS ROMEO AND JULIET ENGLISH — FIRST 20 WORDS

Word	# Occurences	Mean Count	Mean Est. Count (MEC)	Exp. Count	MEC Abs. Err.	MEC Rel Err. (%)	Mean Abs. Err.	Max Abs. Err.	Min Abs. Err.	Mean Rel. Err. (%)	Max Rel. Err. (%)	Min Rel. Err. (%)
tragedy	5003	0.5	1	1	0	0.06	1	1	1	100	100	100
romeo	10000	149	298.01	298	0.01	0	13.87	68	0	4.65	22.82	0
william	5033	0.5	1.01	1	0.01	0.66	1	1	1	100	100	100
shakespeare	4886	0.49	0.98	1	0.02	2.28	1	1	1	100	100	100
contents	5029	0.5	1.01	1	0.01	0.58	1	1	1	100	100	100
prologue	8709	1.48	2.97	3	0.03	1.13	1.51	3	1	50.23	100	33.33
public	9684	2.46	4.93	5	0.07	1.44	1.87	5	1	37.41	100	20
ii	9998	5.97	11.93	12	0.07	0.54	2.68	12	0	22.36	100	0
street	9992	4.99	9.97	10	0.03	0.26	2.46	10	0	24.61	100	0
capulet's	10000	10.99	21.97	22	0.03	0.13	3.67	18	0	16.7	81.82	0
scene	10000	25.45	50.91	51	0.09	0.19	5.7	29	1	11.17	56.86	1.96
v	9991	5.01	10.01	10	0.01	0.14	2.46	10	0	24.56	100	0
house	10000	16.01	32.02	32	0.02	0.05	4.42	20	0	13.8	62.5	0
act	9998	6.51	13.02	13	0.02	0.16	2.95	13	1	22.68	100	7.69
chorus	9856	3.01	6.02	6	0.02	0.39	1.86	6	0	31.01	100	0
place	10000	8.05	16.1	16	0.1	0.63	3.13	14	0	19.55	87.5	0
garden	9964	4.01	8.01	8	0.01	0.18	2.16	8	0	27.05	100	0
vi	7507	1.01	2.01	2	0.01	0.63	1.01	2	0	50.49	100	0
friar	10000	51.91	103.83	104	0.17	0.16	8.17	36	0	7.86	34.62	0
lawrence's	9992	5.01	10.02	10	0.02	0.24	2.48	10	0	24.78	100	0

By analyzing Table VI the following conclusions can be taken:

- Over 10,000 iterations, the mean estimated count (**MEC**) for a word, when rounded to the nearest integer, matches the **expected count** of the Exact Algorithm for the first 20 words. Furthermore, as the probability of each word being counted is $p = \frac{1}{2}$, the **Mean Count** value is exactly/is roughly half of the **MEC**/**expected count** values.
- The **number of occurrences** for a given word is closely related to the exact count of the word and the probability $p = \frac{1}{2}$ used by the algorithm. Therefore, rare words that only appear once such as "tragedy" and "shakespeare" tend to be counted only in roughly half of the iterations — 5003 and 4886, respectively. However, frequently occurring words as romeo are not missed in any of the iterations with **#Occurences** corresponding to 10,000. Furthermore, the fact that it is a probabilistic algorithm also would change the order by which the words appear on this table. Although the presented results include 10,000 iterations the order of the results of Table VI corresponds to the order retrieved by the first run with a seed of 108287. Therefore, if the seed is changed (promote randomness) the order of words of the table would change to the new order retrieved by the first iteration. An experiment with a different seed (for $n\_iter = 1$) will be experimented and discussed later.
- By analyzing columns **Min. Abs. Error** and **Min. Rel. Error** it's possible to conclude that, because $p = \frac{1}{2}$, the algorithm will

never make a right estimate for words that appear an odd number of times. Therefore, words that only appear once have a **Mean, Max and Min Absolute error** and **Mean, Max and Min Relative error** of 1 and 100%, respectively, as they either overestimate or underestimate by 1/100%;

- **Max. Abs. Error** demonstrates that frequently occurring words tend to have an higher maximum absolute error, which is to be expected as it is an absolute value.
- Another more insightful analysis would be that of the **Max. Rel. Error** column that demonstrates that rare words tend to have higher maximum relative errors. On one hand, words that appear a few number of times such as "v" and "tragedy" have a maximum relative error of 100%. On the other hand, words that appear more frequently such as "romeo" have a comparatively low maximum relative error of 22.82%.
- Both of these points hold true for the **Mean. Abs. Error** and **Mean. Rel. Error** — the first tends to be higher for frequently occurring words, while the second tends to be higher for rarely occurring words;
- The metrics **MEC Abs. Err.** and **MEC Rel. Err.** were included as to show that the absolute and relative error values that have been under comparison diminish significantly when performing multiple iterations (replicate to increase certainty). However, these comes at the cost of higher computation resources and memory consumption. Regarding memory consumption this algorithm would allow to use theoretically half the memory of the Exact Counter Algorithm if only one iteration is performed. Therefore, although it is a good idea to use probability to decrease memory consumption, investigation is needed to figure out the kind of results that are retrieved when performing only one iteration, which will be investigated at a later section.

Apart from all the metrics collected and presented on Table VI additional metrics were computed and are presented on Table VII. The mean execution time of 0.00126 (s) was lower than that of the Exact Counter Algorithm for the same book (0.00104 seconds). The pattern observed in the previous table also appears here with the Mean Estimated Total Word Count closely matching the expected number of words of the book (14455.817 and 14456), exhibiting an extremely low relative error of 0.6702%. The min absolute and relative errors considering all words in all iterations was 0.0 and 0.0%, respectively. The max absolute and relative errors are 68 (correspondent to word "romeo") and 100%. All of the values are within what's to be expected for this algorithm.

TABLE VII
ADDITIONAL METRICS OF FIXED PROBABILITY COUNTER OVER 10,000 ITER. — ROMEO AND JULIET ENGLISH

Metric	Value
Execution Time (s)	0.00126
Expected Total Word Count	14456
Mean Total Word Count	7227.9089
Mean Estimated Total Word Count	14455.8178
Mean Relative Error Est. Total Word Count	0.6702%
Max Absolute Error All Words	68.0
Min Absolute Error All Words	0.0
Max Relative Error All Words	100.0%
Min Relative Error All Words	0.0%

On Table VIII, there are presented the top 20 most common words found by the fixed probability algorithm over 10,000 iterations (sorted by the column **MEC**, which is calculated from the mean of all iterations). The 20 most common words and counts found by the Exact Counter Algorithm on Table II are the following: ("romeo", 298), ("thou", 277), ("juliet", 178), ("thy", 170), ("nurse", 146), ("capulet", 141), ("love", 136), ("thee", 135), ("shall", 110), ("lady", 109), ("friar", 104), ("come", 94), ("mercutio", 83), ("good", 80), ("benvolio", 79), ("enter", 75), ("go", 75), ("i'll", 71), ("tybalt", 69) and ("death", 69). Thus, the top 20 most frequent words found by the probabilistic algorithm, as well as their mean estimated counts, matches the ones from the Exact Counter Algorithm (over 10,000 iterations). Furthermore, as this is a list of the most common words it's possible to conclude that no iterations missed any of the words as it's shown on **#Occurences** column. The previously seen trends of the Mean Absolute Error being higher for higher count words, as well as the Mean Relative Error being lower for higher count words are also highlighted on the table.

TABLE VIII
PERFORMANCE METRICS OF FIXED PROBABILITY COUNTER ALGORITHM OVER 10,000 ITERATIONS ROMEO AND JULIET ENGLISH — TOP 20 WORDS (SORTED)

Word	# Occurences	Mean Count	Mean Est. Count (MEC)	Exp. Count	MEC Abs. Err.	MEC Rel Err. (%)	Mean Abs. Err.	Max Abs. Err.	Min Abs. Err.	Mean Rel. Err. (%)	Max Rel. Err. (%)	Min Rel. Err. (%)
romeo	10000	149	298.01	298	0.01	0	13.87	68	0	4.65	22.82	0
thou	10000	138.45	276.9	277	0.1	0.03	13.25	65	1	4.78	23.47	0.36
juliet	10000	89.09	178.18	178	0.18	0.1	10.53	54	0	5.92	30.34	0
thy	10000	84.92	169.84	170	0.16	0.09	10.47	48	0	6.16	28.24	0
nurse	10000	73.03	146.07	146	0.07	0.05	9.56	52	0	6.55	35.62	0
capulet	10000	70.51	141.01	141	0.01	0.01	9.51	45	1	6.74	31.91	0.71
love	10000	68.05	136.11	136	0.11	0.08	9.25	44	0	6.8	32.35	0
thee	10000	67.49	134.99	135	0.01	0.01	9.26	49	1	6.86	36.3	0.74
shall	10000	55.04	110.08	110	0.08	0.07	8.32	48	0	7.56	43.64	0
lady	10000	54.49	108.99	109	0.01	0.01	8.42	39	1	7.73	35.78	0.92
friar	10000	51.91	103.83	104	0.17	0.16	8.17	36	0	7.86	34.62	0
come	10000	47.05	94.11	94	0.11	0.11	7.77	44	0	8.26	46.81	0
mercutio	10000	41.45	82.91	83	0.09	0.11	7.28	37	1	8.77	44.58	1.2
good	10000	40.05	80.1	80	0.1	0.13	7.13	34	0	8.91	42.5	0
benvolio	10000	39.48	78.97	79	0.03	0.04	7.12	39	1	9.02	49.37	1.27
go	10000	37.51	75.01	75	0.01	0.01	6.96	33	1	9.29	44	1.33
enter	10000	37.47	74.93	75	0.07	0.09	6.91	35	1	9.22	46.67	1.33
i'll	10000	35.51	71.01	71	0.01	0.01	6.7	31	1	9.44	43.66	1.41
death	10000	34.5	68.99	69	0.01	0.01	6.7	35	1	9.7	50.72	1.45
tybalt	10000	34.49	68.98	69	0.02	0.03	6.68	31	1	9.68	44.93	1.45

Table IX showcases the 20 least common words found by the fixed probability algorithm over 10,000 iterations (sorted by the column **MEC**, which is calculated from the mean of all iterations). The 20 least common words and counts found by the Exact Counter Algorithm on Table III are the following: ("departed", 1), ("county's", 1), ("rais'd", 1), ("friar's", 1), ("writes", 1), ("'pothecary", 1), ("therewithal", 1), ("scourge", 1), ("winking", 1), ("brace", 1), ("punish'd", 1), ("jointure", 1), ("demand", 1), ("statue", 1), ("whiles", 1), ("figure", 1), ("sacrifices", 1), ("glooming", 1), ("pardon'd", 1) and ("punished", 1). In comparison, the words retrieved by the fixed probability algorithm are aligned with this results with the same value of MEC/Expected Count of roughly 1. However, the list of retrieved words is not exactly the same as it was observed for the 20 most common words. This is due to the probabilistic nature of the algorithm, as well as the way data was collected for the multiple iterations. Another more insightful statistic would be to compare the number of words with an exact count of 1 from the Exact Counter Algorithm to the number of words with an estimated count of 1 from the Fixed Probability Algorithm. As expected, both algorithms yield the same result, 2182 words, therefore showcasing that the algorithm is working correctly (this results can be verified by analyzing the result files that accompany this report). Table IX, in its last 6 columns, also showcases the previously mentioned trend of absolute and relative errors (always 1/100%, respectively) for words with an odd frequency count.

TABLE IX
PERFORMANCE METRICS OF FIXED PROBABILITY COUNTER ALGORITHM OVER 10,000 ITERATIONS ROMEO AND JULIET ENGLISH — BOTTOM 20 WORDS (SORTED)

Word	# Occurences	Mean Count	Mean Est. Count (MEC)	Exp. Count	MEC Abs. Err.	MEC Rel Err. (%)	Mean Abs. Err.	Max Abs. Err.	Min Abs. Err.	Mean Rel. Err. (%)	Max Rel. Err. (%)	Min Rel. Err. (%)
suspected	4907	0.49	0.98	1	0.02	1.86	1	1	1	100	100	100
whiles	4903	0.49	0.98	1	0.02	1.94	1	1	1	100	100	100
kept	4922	0.49	0.98	1	0.02	1.56	1	1	1	100	100	100
ripening	4918	0.49	0.98	1	0.02	1.64	1	1	1	100	100	100
resolution	4914	0.49	0.98	1	0.02	1.72	1	1	1	100	100	100
ambiguities	4904	0.49	0.98	1	0.02	1.92	1	1	1	100	100	100
supple	4911	0.49	0.98	1	0.02	1.78	1	1	1	100	100	100
weaker	4864	0.49	0.97	1	0.03	2.72	1	1	1	100	100	100
retire	4871	0.49	0.97	1	0.03	2.58	1	1	1	100	100	100
waiting	4850	0.48	0.97	1	0.03	3	1	1	1	100	100	100
moveth	4871	0.49	0.97	1	0.03	2.58	1	1	1	100	100	100
forsooth	4868	0.49	0.97	1	0.03	2.64	1	1	1	100	100	100
mousehunt	4858	0.49	0.97	1	0.03	2.84	1	1	1	100	100	100
trunk	4831	0.48	0.97	1	0.03	3.38	1	1	1	100	100	100
fortnight	4874	0.49	0.97	1	0.03	2.52	1	1	1	100	100	100
world—why	4872	0.49	0.97	1	0.03	2.56	1	1	1	100	100	100
justly	4866	0.49	0.97	1	0.03	2.68	1	1	1	100	100	100
stint	4862	0.49	0.97	1	0.03	2.76	1	1	1	100	100	100
quit	4861	0.49	0.97	1	0.03	2.78	1	1	1	100	100	100
enforce	4820	0.48	0.96	1	0.04	3.6	1	1	1	100	100	100

B.2 Memory Usage Analysis - 1 Iteration

Similarly to the experiments done on the Exact Counter Algorithm, the memory consumption throughout the processing of each word has also been tracked for the Fixed Probability Algorithm (only for 1 iteration). This information is presented on Table X. On Table X it's clear the probabilistic nature of the algorithm where words such as "william", "prologue" and "public" were not counted the first time they appeared, therefore not affecting neither the memory usage nor the number of distinct words. This also happens for instance the second and third time the word "scene" appears. On the last 5 lines of the table (corresponding to the last 5 processed words by the algorithm), the key takeaway is that the memory usage column exhibits lower values (224136 bytes) comparatively to the ones from the Exact Counter Algorithm (383016 bytes). Furthermore, the number of distinct words is also lower (2419 instead of 3734) which is also to be expected given that theoretically only half of the words (not distinct words) are counted.

TABLE X
MEMORY USAGE BY WORD STATISTICS ROMEO AND JULIET ENGLISH — FIXED PROB. ALG.

Current Word	Current Word Count	Memory Usage	# Distinct Words
		64	
tragedy	1	320	1
romeo	1	376	2
juliet	1	432	3
william	0	432	3
shakespeare	1	496	4
contents	1	560	5
prologue	0	560	5
act	1	744	6
scene	1	800	7
public	0	800	7
place	0	800	7
scene	1	800	7
ii	1	856	8
street	1	912	9
scene	1	912	9
iii	1	968	10
room	1	1304	11
capulet's	0	1304	11
house	1	1360	12
...	...	...	...
scene	1	1560	15
open	1	1616	16
place	1	1672	17
...	...	...	...
story	1	224136	2419
woe	7	224136	2419
juliet	92	224136	2419
romeo	161	224136	2419
exeunt	12	224136	2419

Figure 4 plots the behavior mentioned in the previous table. Comparatively to the memory consumption shown in Figure 1 for the Exact Counter Algorithm it shares the linear patterns and breaks in continuity related to memory allocations. However, the memory consumption grows at a lower rate as it's possible to see by the order of magnitude of the y axis which is roughly half. This behavior is also present for the german and french books, whose plots will not be presented on the report, but can be found on the result files that accompany it.

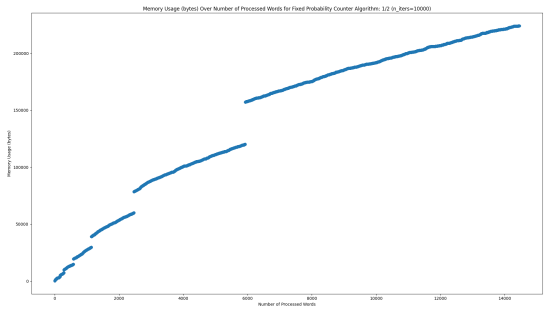


Fig. 4 - Memory Usage (B) by Number of Processed Words for Fixed Probability Algorithm — Romeo and Juliet English

B.3 Word Frequency Analysis - 1 Iteration, seed 108288

In the last section the behavior of the Fixed Probability Algorithm ($p = \frac{1}{2}$) over 10,000 iterations has been analyzed, yielding impressive results. However, a more insightful analysis may be observing the same algorithmic metrics for a single iteration. For the sake of promoting randomness and observing different results the initial seed for the Fixed Probability Algorithm has been changed from 108287 to 108288.

Table XI presents the already seen metrics, for the Fixed Probability Algorithm, with seed 108288 and $n\_iters = 1$. Due to the fact that only 1 iteration is being made the values for the **MEC Abs. Err.** and **MEC Rel. Err.** columns coincide with the ones for the **Mean Abs. Err.** and **Mean Rel. Err.** columns, respectively, showing that the metrics are being well calculated. Based on the data presented in Table XI, the following conclusions can be made:

- Due to the different seed of 108288 used for this iteration the order of the first 20 words identified by the algorithm is not the same as the one identified in Table VI. This is expected due to the probabilistic nature of the algorithm, as each word has always a 50% ($p = \frac{1}{2}$) chance of being processed;
- **#Occurrences** column has always the value of 1 and the **Mean Count** value is always an integer number, as only 1 iteration has been performed;
- The mean estimated count (**MEC**) only matched the expected (exact) count 5 times for the first 20 words. This highlights a drop in accuracy for only one instance of the algorithm due to its probabilistic nature and volatility (multiple repetitions are not performed to increase certainty). Therefore, the **MEC Abs. Err.** and **MEC Rel. Err.** values are also generally (excluding the times when it's exactly 0) way worse for 1 iteration than for the previously seen values for 10,000 iterations.

TABLE XI
PERFORMANCE METRICS OF FIXED PROBABILITY COUNTER ALGORITHM OVER 1 ITERATION — SEED 108288 ROMEO AND JULIET ENGLISH — FIRST 20 WORDS

Word	# Occurrences	Mean Count	Mean Est. Count (MEC)	Exp. Count	MEC Abs. Err.	MEC Rel Err. (%)	Mean Abs. Err.	Max Abs. Err.	Min Abs. Err.	Mean Rel. Err. (%)	Max Rel. Err. (%)	Min Rel. Err. (%)
contents	1	1	2	1	1	100	1	1	1	100	100	100
prologue	1	2	4	3	1	33.33	1	1	1	33.33	33.33	33.33
act	1	6	12	13	1	7.69	1	1	1	7.69	7.69	7.69
scene	1	29	58	51	7	13.73	7	7	7	13.73	13.73	13.73
public	1	3	6	5	1	20	1	1	1	20	20	20
place	1	11	22	16	6	37.5	6	6	6	37.5	37.5	37.5
street	1	5	10	10	0	0	0	0	0	0	0	0
iii	1	6	12	12	0	0	0	0	0	0	0	0
room	1	4	8	8	0	0	0	0	0	0	0	0
capulet's	1	12	24	22	2	9.09	2	2	2	9.09	9.09	9.09
iv	1	4	8	10	2	20	2	2	2	20	20	20
hall	1	3	6	8	2	25	2	2	2	25	25	25
house	1	13	26	32	6	18.75	6	6	6	18.75	18.75	18.75
ii	1	6	12	12	0	0	0	0	0	0	0	0
chorus	1	2	4	6	2	33.33	2	2	2	33.33	33.33	33.33
adjoining	1	2	4	2	2	100	2	2	2	100	100	100
garden	1	4	8	8	0	0	0	0	0	0	0	0
vi	1	2	4	2	2	100	2	2	2	100	100	100
friar	1	41	82	104	22	21.15	22	22	22	21.15	21.15	21.15
lawrence's	1	5	10	10	0	0	0	0	0	0	0	0

Table XII shows additional metrics for the same seed and number of iterations. The Execution Time is within what's to be expected close to the mean value that was previously seen (0.00126 seconds). It's also possible to conclude that for one iteration the algorithm does not provide such an accurate Mean Estimated Total Word Count with the final result being 14488.0, which is an overestimate of 32 words. Nevertheless, it still constitutes a close value that is roughly half of the estimated count.

TABLE XII
ADDITIONAL METRICS OF FIXED PROBABILITY COUNTER ALGORITHM WITH 1 ITERATION & SEED 108288 ROMEO AND JULIET ENGLISH

Metric	Value
Execution Time (s)	0.00129
Expected Total Word Count	14456
Mean Total Word Count	7244.0
Mean Estimated Total Word Count	14488.0
Mean Relative Error Est. Total Word Count	0.2214%
Max Absolute Error All Words	32.0
Min Absolute Error All Words	0.0
Max Relative Error All Words	100.0%
Min Relative Error All Words	0.0%

Table XIII illustrates the 20 most common words estimated by the algorithm, along with the respective metrics. For comparison, the 20 most common words and counts found by the Exact Counter Algorithm on Table II are the following: ("romeo", 298), ("thou", 277), ("juliet", 178), ("thy", 170), ("nurse", 146), ("capulet", 141), ("love", 136), ("thee", 135), ("shall", 110), ("lady", 109), ("friar", 104), ("come", 94), ("mercutio", 83), ("good", 80), ("benvolio", 79), ("enter", 75), ("go", 75), ("i'll", 71), ("tybalt", 69) and ("death", 69). This way the fixed probability algorithm only matched this exact ordering of the top 20 words 6 times. However, 18 words from the top 20 retrieved by the Exact Counter Algo-

rithm are present in the Top 20 estimated by the Fixed Probability Algorithm with this seed for 1 iteration, although not in the exact relative order (the 2 words of the exact top 20 that were not present are ranked 21 and 22 of the full estimation list). From the 20 most common words estimated by the algorithm the following conclusions can be made:

- Only once the estimated count matched the exact count, with Absolute and Relative errors equal to zero;
- The previously mentioned trend of the **MEC Abs. Err.** and **MEC Rel. Err.** values being generally way worse for 1 iteration can also be seen here;
- Compared to Table VIII (for 10,000 iterations), the previously seen trend of the Mean Absolute Error being higher for higher count words and the Mean Relative Error being lower for higher count words is not present in this table, which showcases the more probabilistic and erratic nature of the algorithm for one sole iteration with more variable values for the **Mean. Abs. Err.** and **Mean Rel. Err.** columns.

TABLE XIII
PERFORMANCE METRICS OF FIXED PROBABILITY COUNTER ALGORITHM OVER 1 ITERATION — SEED 108288 ROMEO AND JULIET ENGLISH — TOP 20 WORDS (SORTED)

Word	# Occurences	Mean Count	Mean Est. Count (MEC)	Exp. Count	MEC Abs. Err.	MEC Rel Err. (%)	Mean Abs. Err.	Max Abs. Err.	Min Abs. Err.	Mean Rel. Err. (%)	Max Rel. Err. (%)	Min Rel. Err. (%)
thou	1	141	282	277	5	1.81	5	5	5	1.81	1.81	1.81
romeo	1	133	266	298	32	10.74	32	32	32	10.74	10.74	10.74
juliet	1	99	198	178	20	11.24	20	20	20	11.24	11.24	11.24
thy	1	87	174	170	4	2.35	4	4	4	2.35	2.35	2.35
capulet	1	82	164	141	23	16.31	23	23	23	16.31	16.31	16.31
thee	1	75	150	135	15	11.11	15	15	15	11.11	11.11	11.11
nurse	1	74	148	146	2	1.37	2	2	2	1.37	1.37	1.37
love	1	73	146	136	10	7.35	10	10	10	7.35	7.35	7.35
shall	1	57	114	110	4	3.64	4	4	4	3.64	3.64	3.64
come	1	53	106	94	12	12.77	12	12	12	12.77	12.77	12.77
mercutio	1	49	98	83	15	18.07	15	15	15	18.07	18.07	18.07
lady	1	49	98	109	11	10.09	11	11	11	10.09	10.09	10.09
benvolio	1	45	90	79	11	13.92	11	11	11	13.92	13.92	13.92
friar	1	41	82	104	22	21.15	22	22	22	21.15	21.15	21.15
night	1	41	82	68	14	20.59	14	14	14	20.59	20.59	20.59
enter	1	40	80	75	5	6.67	5	5	5	6.67	6.67	6.67
go	1	40	80	75	5	6.67	5	5	5	6.67	6.67	6.67
good	1	40	80	80	0	0	0	0	0	0	0	0
tybalt	1	39	78	69	9	13.04	9	9	9	13.04	13.04	13.04
man	1	37	74	65	9	13.85	9	9	9	13.85	13.85	13.85

Table XIV presents the same metrics, but instead for the bottom 20 words. The 20 least common words and counts found by the Exact Counter Algorithm on Table III are the following: ("departed", 1), ("county's", 1), ("rais'd", 1), ("friar's", 1), ("writes", 1), ("'pothecary", 1), ("therewithal", 1), ("scourge", 1), ("winking", 1), ("brace", 1), ("punish'd", 1), ("jointure", 1), ("demand", 1), ("statue", 1), ("whiles", 1), ("figure", 1), ("sacrifices", 1), ("glooming", 1), ("pardon'd", 1) and ("punished", 1). From this list of bottom 20 least frequent words, which all have an exact count of 1, 8 words were found on the estimated list provided by the fixed probability algorithm, which represents a commendable outcome. One important observation is that because the probability p of the algorithm is $\frac{1}{2}$ the algorithm cannot estimate for a word count of 1. Moreover, the algorithm will estimate a word count of 2 both for words that appear once and were counted once and words that appeared twice and were counted half of the times (and so on). This way, the algorithm estimates a list of 1553 words with MEC 2 as the least frequent words. Therefore, this value doesn't correspond to roughly half of the number of words with exact count value of 1 (2182), providing an overestimate. This is also verified by the results of Table XIV with 6 words with expected count (exact count) of more than 1 being included in the bottom 20 list.

Despite of this overestimation, from a quick analysis from the full result's file it's possible to conclude that for the bottom 20 words (with count 1) retrieved by the Exact Counter Algorithm 11 words (slightly more than half) are not present in the estimate 20 least frequent words of the Fixed Probability Algorithm: "departed", "county's", "friar's", "'pothecary", "winking", "brace", "punish'd", "statue", "figure", "glooming" and "pardon'd".

TABLE XIV
PERFORMANCE METRICS OF FIXED PROBABILITY COUNTER ALGORITHM OVER 1 ITERATION — SEED 108288 ROMEO AND JULIET ENGLISH — BOTTOM 20 WORDS (SORTED)

Word	# Occurences	Mean Count	Mean Est. Count (MEC)	Exp. Count	MEC Abs. Err.	MEC Rel Err. (%)	Mean Abs. Err.	Max Abs. Err.	Min Abs. Err.	Mean Rel. Err. (%)	Max Rel. Err. (%)	Min Rel. Err. (%)
return'd	1	1	2	1	1	100	1	1	1	100	100	100
prefixed	1	1	2	1	1	100	1	1	1	100	100	100
kindred's	1	1	2	2	0	0	0	0	0	0	0	0
meaning	1	1	2	3	1	33.33	1	1	1	33.33	33.33	33.33
closely	1	1	2	1	1	100	1	1	1	100	100	100
conveniently	1	1	2	1	1	100	1	1	1	100	100	100
awaking	1	1	2	1	1	100	1	1	1	100	100	100
seems	1	1	2	2	0	0	0	0	0	0	0	0
miscarried	1	1	2	1	1	100	1	1	1	100	100	100
rais'd	1	1	2	1	1	100	1	1	1	100	100	100
ope	1	1	2	2	0	0	0	0	0	0	0	0
writes	1	1	2	1	1	100	1	1	1	100	100	100
therewithal	1	1	2	1	1	100	1	1	1	100	100	100
scourge	1	1	2	1	1	100	1	1	1	100	100	100
kinsmen	1	1	2	4	2	50	2	2	2	50	50	50
jointure	1	1	2	1	1	100	1	1	1	100	100	100
demand	1	1	2	1	1	100	1	1	1	100	100	100
whiles	1	1	2	1	1	100	1	1	1	100	100	100
sacrifices	1	1	2	1	1	100	1	1	1	100	100	100
enmity	1	1	2	2	0	0	0	0	0	0	0	0

B.4 Memory Usage Analysis - 1 Iteration, seed 108288

Table XV presents the memory consumption of the algorithm as each word is processed. Com-

pared to the previous Table X for seed 108287, Table XV for seed 108288, shows that the different seed impacts when the counter is incremented (and the words for which it is incremented) and, therefore, when more memory is required. The first example where this happens is upon processing of word "contents", which is not counted for seed 108288 and therefore not consuming more memory keeping the memory consumption at 496 bytes, but is counted for seed 108287 incrementing it to 560 bytes. This is a common pattern throughout the lifecycle of the algorithm, which makes the memory usage and the number of distinct words identified be slightly different for different runs/seeds. For example, the algorithm presented on Table XV ends the processing of the last word with memory usage of 227704 bytes and 2470 distinct words, while the one previously observed on Table X ends it with 224136 bytes of memory usage and 2419 distinct words.

TABLE XV
MEMORY USAGE BY WORD STATISTICS OF FIXED PROB. ALG. WITH 1 ITERATION & SEED 108288 ROMEO AND JULIET ENGLISH

Current Word	Current Word Count	Memory Usage	# Distinct Words
		64	
tragedy	1	320	1
romeo	1	376	2
juliet	1	432	3
william	0	432	3
shakespeare	1	496	4
contents	0	496	4
prologue	1	560	5
act	1	744	6
scene	1	800	7
public	1	856	8
place	0	856	8
scene	1	856	8
ii	1	912	9
street	0	912	9
scene	1	912	9
iii	1	968	10
room	1	1304	11
capulet's	0	1304	11
house	1	1360	12
...	...	...	...
scene	2	1712	17
open	0	1712	17
place	1	1768	18
...	...	...	...
story	2	227704	2470
woe	7	227704	2470
juliet	87	227704	2470
romeo	148	227704	2470
exeunt	16	227704	2470

C. Lossy-Count Algorithm

For the analysis of the Lossy-Count Algorithm experiments were made to **estimate the n most frequent words** for the book, with $n \in \{5, 10, 15, 20, 25, 30\}$. These experiments were repeated for multiple values of k ($\frac{1}{\epsilon}$), with $k \in \{100, 500, 1000, 2000, 3000, 14456\}$ — 14456 being the total number of words of the english book. Of all these experiments the ones with $k \in \{100, 500, 3000, 14456\}$ will be discussed in this report. For all of the experiments it was asserted that the algorithm did not overestimate any word count by no more than $\frac{n}{k} = \epsilon \cdot n$, with n being the total number of words of the book.

C.1 Word Frequency Analysis - k = 100

Table XVI presents the top 30 words found by the Lossy Count algorithm, for $k = 100$. Bear in mind that column 3 (Freq. Word Exact Count) has the expected count for the found word by the Lossy Count Algorithm that is present in column 1 (Freq. Word), while the last two columns have the expected word and exact count of the Exact Counter Algorithm for comparison.

TABLE XVI
PERFORMANCE METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH — TOP 30 WORDS (K=100)

Freq. Word	Bucket Value	Freq. Word Exact Count	Abs. Err.	Rel. Err.(%)	Expected Word	Exact Count
romeo	301	298	3	1.01	romeo	298
thou	280	277	3	1.08	thou	277
juliet	198	178	20	11.24	juliet	178
thy	195	170	25	14.71	thy	170
capulet	178	141	37	26.24	nurse	146
nurse	168	146	22	15.07	capulet	141
friar	161	104	57	54.81	love	136
thee	157	135	22	16.3	thee	135
watch	153	23	130	565.22	shall	110
prince	152	37	115	310.81	lady	109
lady	150	109	41	37.61	friar	104
death	149	69	80	115.94	come	94
lawrence	148	67	81	120.9	mercutio	83
montague	148	45	103	228.89	good	80
letter	147	14	133	950	benvolio	79
came	147	13	134	1030.77	enter	75
give	147	47	100	212.77	go	75
shall	147	110	37	33.64	i'll	71
paris	146	58	88	151.72	tybalt	69
come	146	94	52	55.32	death	69
dead	146	47	99	210.64	night	68
master	146	10	136	1360	lawrence	67
love	145	136	9	6.62	man	65
romeo's	145	16	129	806.25	hath	64
time	145	32	113	353.12	one	60
vault	145	9	136	1511.11	paris	58
place	145	16	129	806.25	well	58
bid	145	21	124	590.48	sir	57
page	145	12	133	1108.33	art	55
lady's	145	8	137	1712.5	would	54

From Table XVI it's possible to conclude that the Lossy-Count algorithm, for $k = 100$, found the most common 4 words correctly (columns **Freq. Word** and **Expected Word** match): "romeo", "thou", "juliet" and "thy". Other words, such as "capulet", "nurse", and "friar", although present in the actual top30, show slight variations in their order of appearance in the top ranking. Another important deduction is that the Lossy Count Algorithm only performed overestimates, with the top 2 words showing the minimum absolute and relative errors. However, the relative errors for the remaining words were significantly high and displayed considerable variation.

Table XVII presents the Mean Absolute and Relative errors for the estimates of the **n most frequent words**, with $n \in \{5, 10, 15, 20, 25, 30\}$, as well as some additional metrics. The first conclusion to make is that the algorithm is performing correctly as the experimental value for the maximum overestimate is not more than the theoretical value ($\frac{n}{k}$). It's also possible to verify that the mean absolute and relative errors increase drastically as n (Top n) increases. Although the min and max absolute and relative errors are not present in Table XVII, they were computed and

included in the project result files and the following conclusions can be made:

- The value of the **Max Absolute Error** increased as n increased;
- The **Min Absolute Error** remained consistent as n increased always corresponding to the value of the Min Absolute Error of the Top 5 words;
- The value of the **Max Relative Error** increased drastically as n increased;
- The **Min Relative Error** remained consistent as n increased always corresponding to the value of the Min Relative Error of the Top 5 words.

TABLE XVII
ADDITIONAL METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH (K=100)

Metric	Value
Execution Time (s)	0.00434
Theoretical Max Overestimation Error ($\frac{n}{k}$)	144.56
Final Delta Δ	144
Actual Max Overestimation Error	144
Theoretical >= Actual Max Overestimation Error	True
Mean Absolute Error Top 5	17.6
Mean Relative Error Top 5	10.85%
Mean Absolute Error Top 10	43.4
Mean Relative Error Top 10	101.65%
Mean Absolute Error Top 15	58.13
Mean Relative Error Top 15	164.65%
Mean Absolute Error Top 20	64.15
Mean Relative Error Top 20	197.70%
Mean Absolute Error Top 25	70.76
Mean Relative Error Top 25	267.63%
Mean Absolute Error Top 30	80.93
Mean Relative Error Top 30	413.98%

C.2 Memory Usage Analysis - k = 100

TABLE XVIII
MEMORY USAGE BY WORD STATISTICS OF LOSSY-COUNT ROMEO AND JULIET ENGLISH (K=100)

Word	isNewWord	Word Count	Delta	Memory Usage	#Buckets
			0	64	0
tragedy	True	1	0	320	1
romeo	True	1	0	376	2
juliet	True	1	0	432	3
william	True	1	0	488	4
shakespeare	True	1	0	552	5
contents	True	1	0	744	6
prologue	True	1	0	808	7
act	True	1	0	864	8
scene	True	1	0	920	9
public	True	1	0	976	10
place	True	1	0	1312	11
scene	False	2	0	1344	11
ii	True	1	0	1400	12
street	True	1	0	1456	13
scene	False	3	0	1456	13
iii	True	1	0	1512	14
room	True	1	0	1568	15
capulet's	True	1	0	1664	16
house	True	1	0	1720	17
...	...	...	...	...	...
miss	True	3	2	13584	143
toil	True	3	2	13640	144
shall	False	4	2	13640	144
strive	True	3	2	13696	145
mend	True	3	2	13752	146
exit	True	3	3	12216	121
...	...	...	...	...	...
juliet	False	198	144	18264	141
romeo	False	301	144	18264	141
exeunt	True	145	144	18320	142

Table XVIII highlights the memory consumption of the Lossy-Count algorithm as each word is processed. As previously mentioned, the **memory usage** (and also **#Buckets**) doesn't change due to incrementing a counter for a pre-existing word such as "shall". Nevertheless, the key takeaways to take from the Table are the following:

- The algorithm actively performs a pruning step to remove infrequent elements, diminishing **memory consumption** and the **#Buckets**. This can be verified on lines 26 and 27 of Table XVIII;
- The delta value Δ, corresponding to the maximum possible error in the frequency of the word, evolves from 0 until 144, which was previously shown on Table XVII;
- From the analysis of the full table in the project result's files it's possible to conclude that the Maximum Number of Buckets used during the run was 197. Clearly, pruning operations have been carried out as the algorithm ends with 142 active buckets.

Figure 5 plots the memory consumption in bytes over the number of processed words for the Lossy Count Algorithm, with k=100. The plot effectively highlights the impact on memory consumption of pruning words whose frequencies fall below a given delta Δ at the given step. It's also important to highlight the order of magnitude of the y-axis for memory consumption (high of 30000 bytes), which is way lower than the previously seen values of 400000 (Exact Counter Algorithm y-axis) and 200000 (Fixed Probability Algorithm y-axis). This behavior is expected, given the algorithm's design and the paradigm shift compared to previous approaches. At last, this Figure is also important to notify that the 18320 byte value on the last line of Table XVIII (memory usage value after processing the last word) is not the highest memory usage value throughout the lifecycle of the data stream algorithm.

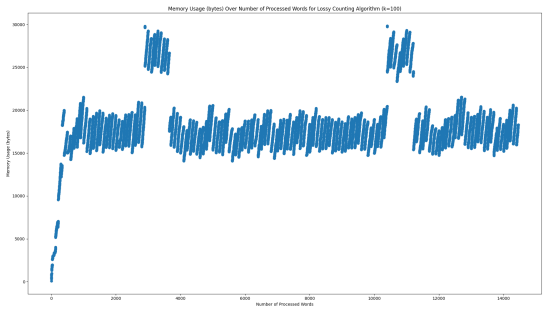


Fig. 5 - Memory Usage (B) by Number of Processed Words for Lossy Count Algorithm — Romeo and Juliet English (k=100)

C.3 Word Frequency Analysis - k = 500

After changing the value of k to 500, Table XIX shows that the algorithm correctly identifies the Top 15 words (although this are not all the words whose count are estimated correctly or in the same

order as the exact algorithm). The values of the Absolute and Relative Errors also decreased significantly compared to $k = 100$, with the word with the worst performance being 13.79% off the correct value and existing a lot of words whose estimated count closely matched the respective exact value count (more common among the most common words at the top of the table).

TABLE XIX
PERFORMANCE METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH — TOP 30 WORDS (K=500)

Freq. Word	Bucket Value	Freq. Word Exact Count	Abs. Err.	Rel. Err.(%)	Expected Word	Exact Count
romeo	298	298	0	0	romeo	298
thou	277	277	0	0	thou	277
juliet	178	178	0	0	juliet	178
thy	170	170	0	0	thy	170
nurse	147	146	1	0.68	nurse	146
capulet	141	141	0	0	capulet	141
love	137	136	1	0.74	love	136
thee	135	135	0	0	thee	135
shall	110	110	0	0	shall	110
lady	109	109	0	0	lady	109
friar	106	104	2	1.92	friar	104
come	95	94	1	1.06	come	94
mercutio	85	83	2	2.41	mercutio	83
good	81	80	1	1.25	good	80
benvolio	79	79	0	0	benvolio	79
go	76	75	1	1.33	enter	75
enter	75	75	0	0	go	75
lawrence	75	67	8	11.94	i'll	71
death	74	69	5	7.25	tybalt	69
i'll	72	71	1	1.41	death	69
tybalt	69	69	0	0	night	68
night	69	68	1	1.47	lawrence	67
paris	66	58	8	13.79	man	65
man	65	65	0	0	hath	64
hath	65	64	1	1.56	one	60
one	60	60	0	0	paris	58
well	58	58	0	0	well	58
sir	57	57	0	0	sir	57
would	57	54	3	5.56	art	55
art	56	55	1	1.82	would	54

As previously, Table XX presents the Mean Absolute and Relative errors for the estimates of the n most frequent words, with $n \in 5, 10, 15, 20, 25, 30$, but for $k = 500$. Similarly to Table XVII, the experimental value for the maximum overestimate is not more than the theoretical value ($\frac{n}{k}$). However, although the mean absolute and relative errors tend to increase as n increases it's not so noticeable within the top 30 anymore, but in later iterations of n (which are not shown in order not to extend the size of this report further).

TABLE XX
ADDITIONAL METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH (K=500)

Metric	Value
Execution Time (s)	0.00402
Theoretical Max Overestimation Error ($\frac{n}{k}$)	28.912
Final Delta Δ	28
Actual Max Overestimation Error	28
Theoretical >= Actual Max Overestimation Error	True
Mean Absolute Error Top 5	0.2
Mean Relative Error Top 5	0.137%
Mean Absolute Error Top 10	0.2
Mean Relative Error Top 10	0.142%
Mean Absolute Error Top 15	0.53
Mean Relative Error Top 15	0.538%
Mean Absolute Error Top 20	1.15
Mean Relative Error Top 20	1.500%
Mean Absolute Error Top 25	1.32
Mean Relative Error Top 25	1.873%
Mean Absolute Error Top 30	1.23
Mean Relative Error Top 30	1.807%

C.4 Memory Usage Analysis - k = 500

Table XXI and Figure 6 showcase the memory consumption (B) over the number of processed words for the Lossy Count Algorithm ($k = 500$).

Comparatively to the last memory usage table for $k = 100$ the delta value Δ, corresponding to the maximum possible error in the frequency of the word, evolves at a slower rate for $k = 500$. Therefore, at the time of the insertion of the "exit" word the memory consumption is higher than for $k = 100$ as pruning as not yet been performed. Due to the reduced pruning operations, the overall number of buckets used throughout the algorithm lifecycle is also greater with a maximum number of buckets of 712 (which is also higher than the 197 observed for $k = 100$). This behavior is expected as k increases.

TABLE XXI
MEMORY USAGE BY WORD STATISTICS OF LOSSY-COUNT ROMEO AND JULIET ENGLISH (K=500)

Word	isNewWord	Word Count	Delta	Memory Usage	#Buckets
			0	64	0
tragedy	True	1	0	320	1
romeo	True	1	0	376	2
juliet	True	1	0	432	3
william	True	1	0	488	4
shakespeare	True	1	0	552	5
contents	True	1	0	744	6
prologue	True	1	0	808	7
act	True	1	0	864	8
scene	True	1	0	920	9
public	True	1	0	976	10
place	True	1	0	1312	11
scene	False	2	0	1344	11
ii	True	1	0	1400	12
street	True	1	0	1456	13
scene	False	3	0	1456	13
iii	True	1	0	1512	14
room	True	1	0	1568	15
capulet's	True	1	0	1664	16
house	True	1	0	1720	17
...	...	...	...	...	...
miss	True	1	0	14064	151
toil	True	1	0	14120	152
shall	False	2	0	14120	152
strive	True	1	0	14176	153
mend	True	1	0	14232	154
exit	True	1	0	14288	155
...	...	...	...	...	...
juliet	False	178	28	75000	607
romeo	False	298	28	75000	607
exeunt	False	35	28	75000	607

Figure 6 exhibits a similar pattern to the one previously observed due to the pruning operations. However, these pruning operations happen at an higher order of magnitude of 80000 bytes for the higher k value.

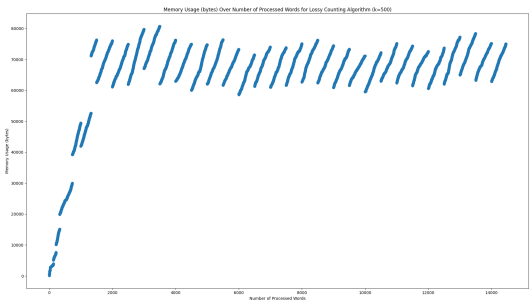


Fig. 6 - Memory Usage (B) by Number of Processed Words for Lossy Count Algorithm — Romeo and Juliet English (k=500)

C.5 Word Frequency Analysis - $k = 3000$

For $k = 3000$, the Lossy-Count Algorithm perfectly matches the top 30 words exactly as the Exact Counter Algorithm with absolute and relative errors corresponding to zero, as shown in Table XXII.

TABLE XXII
PERFORMANCE METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH — TOP 30 WORDS (K=3000)

Freq. Word	Bucket Value	Freq. Word Exact Count	Abs. Err.	Rel. Err.(%)	Expected Word	Exact Count
romeo	298	298	0	0	romeo	298
thou	277	277	0	0	thou	277
juliet	178	178	0	0	juliet	178
thy	170	170	0	0	thy	170
nurse	146	146	0	0	nurse	146
capulet	141	141	0	0	capulet	141
love	136	136	0	0	love	136
thee	135	135	0	0	thee	135
shall	110	110	0	0	shall	110
lady	109	109	0	0	lady	109
friar	104	104	0	0	friar	104
come	94	94	0	0	come	94
mercutio	83	83	0	0	mercutio	83
good	80	80	0	0	good	80
benvolio	79	79	0	0	benvolio	79
enter	75	75	0	0	enter	75
go	75	75	0	0	go	75
i'll	71	71	0	0	i'll	71
tybalt	69	69	0	0	tybalt	69
death	69	69	0	0	death	69
night	68	68	0	0	night	68
lawrence	67	67	0	0	lawrence	67
man	65	65	0	0	man	65
hath	64	64	0	0	hath	64
one	60	60	0	0	one	60
paris	58	58	0	0	paris	58
well	58	58	0	0	well	58
sir	57	57	0	0	sir	57
art	55	55	0	0	art	55
would	54	54	0	0	would	54

Table XXIII reflects what was concluded on the previous table with mean absolute and relative errors of 0.0. Although, as k increases the algorithm starts behaving more and more like the exact counter algorithm, as we'll later see, $k = 3000$ is not sufficient for the algorithm to perfectly "estimate" every single word of the book, with a mean absolute error of 2.1785.

TABLE XXIII
ADDITIONAL METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH (K=3000)

Metric	Value
Execution Time (s)	0.00373
Theoretical Max Overestimation Error ($\frac{n}{k}$)	4.819
Final Delta Δ	4
Actual Max Overestimation Error	4
Theoretical >= Actual Max Overestimation Error	True
Mean Absolute Error Top 5	0.0
Mean Relative Error Top 5	0.0%
Mean Absolute Error Top 10	0.0
Mean Relative Error Top 10	0.0%
Mean Absolute Error Top 15	0.0
Mean Relative Error Top 15	0.0%
Mean Absolute Error Top 20	0.0
Mean Relative Error Top 20	0.0%
Mean Absolute Error Top 25	0.0
Mean Relative Error Top 25	0.0%
Mean Absolute Error Top 30	0.0
Mean Relative Error Top 30	0.0%

C.6 Memory Usage Analysis - $k = 3000$

As k increases the behavior of the Lossy-Count algorithm starts becoming like the Exact Counter Algorithm due to the high number of buckets and small number of pruning operations. In Table XXIV it's possible to verify the really high number of buckets and memory consumption with a maximum of 2227 buckets and 285992 bytes, respectively (both increased drastically from $k = 500$). However, 2227 buckets it's still not sufficient to accommodate an exact count for the 3734 distinct words present in the book resulting in the previously mentioned absolute error of 2.1785.

TABLE XXIV
MEMORY USAGE BY WORD STATISTICS OF LOSSY-COUNT ROMEO AND JULIET ENGLISH (K=3000)

Word	isNewWord	Word Count	Delta	Memory Usage	#Buckets
			0	64	0
tragedy	True	1	0	320	1
romeo	True	1	0	376	2
juliet	True	1	0	432	3
william	True	1	0	488	4
shakespeare	True	1	0	552	5
contents	True	1	0	744	6
prologue	True	1	0	808	7
act	True	1	0	864	8
scene	True	1	0	920	9
public	True	1	0	976	10
place	True	1	0	1312	11
scene	False	2	0	1344	11
ii	True	1	0	1400	12
street	True	1	0	1456	13
scene	False	3	0	1456	13
iii	True	1	0	1512	14
room	True	1	0	1568	15
capulet's	True	1	0	1664	16
house	True	1	0	1720	17
...	...	...	...	...	...
digressing	True	3	2	285936	2226
valour	True	3	2	285992	2227
man	False	46	2	285992	2227
thy	False	101	3	238840	1473
...	...	...	...	...	...
story	True	5	4	274168	2022
woe	False	13	4	274168	2022
juliet	False	178	4	274168	2022
romeo	False	298	4	274168	2022
exeunt	False	34	4	274168	2022

Figure 7 showcases an even higher order of magnitude for the memory usage y-axis. It also shows a pattern that's becoming more similar to the one observed on the Exact Count Algorithm instead of the ones observed for the Lossy-Count for other k's.

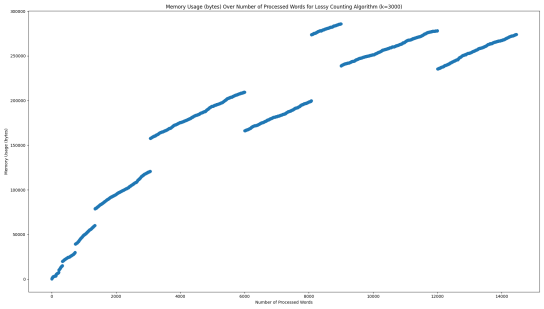


Fig. 7 - Memory Usage (B) by Number of Processed Words for Lossy Count Algorithm — Romeo and Juliet English (k=3000)

C.7 Word Frequency Analysis - $k = 14456$

In this section, a value of k equal to the number of words in the book (14456) is used in order to guarantee an exact estimate of word count for all words and showcase that for this value of k no pruning operations are performed and the number of buckets is sufficiently high for the algorithm to

behave exactly as the Exact Counter Algorithm, and therefore providing an exact estimate. Table XXV exhibits the same behavior as for $k = 3000$ giving an exact estimate for all words in the Top 30.

TABLE XXV
PERFORMANCE METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH — TOP 30 WORDS (K=14456)

Freq. Word	Bucket Value	Freq. Word Exact Count	Abs. Err.	Rel. Err.(%)	Expected Word	Exact Count
romeo	298	298	0	0	romeo	298
thou	277	277	0	0	thou	277
juliet	178	178	0	0	juliet	178
thy	170	170	0	0	thy	170
nurse	146	146	0	0	nurse	146
capulet	141	141	0	0	capulet	141
love	136	136	0	0	love	136
thee	135	135	0	0	thee	135
shall	110	110	0	0	shall	110
lady	109	109	0	0	lady	109
friar	104	104	0	0	friar	104
come	94	94	0	0	come	94
mercutio	83	83	0	0	mercutio	83
good	80	80	0	0	good	80
benvolio	79	79	0	0	benvolio	79
enter	75	75	0	0	enter	75
go	75	75	0	0	go	75
i'll	71	71	0	0	i'll	71
tybalt	69	69	0	0	tybalt	69
death	69	69	0	0	death	69
night	68	68	0	0	night	68
lawrence	67	67	0	0	lawrence	67
man	65	65	0	0	man	65
hath	64	64	0	0	hath	64
one	60	60	0	0	one	60
paris	58	58	0	0	paris	58
well	58	58	0	0	well	58
sir	57	57	0	0	sir	57
art	55	55	0	0	art	55
would	54	54	0	0	would	54

Relatively to Table XXVI it is shown the theoretical max overestimate value of 1.0, as well as the null metrics for the mean absolute and relative errors for Top 30. What is not shown in this table is the Mean Absolute and Relative errors for the full list of words which is exactly 0.0 and 0.0%, respectively, therefore showcasing the exact behavior that has been mentioned.

TABLE XXVI
ADDITIONAL METRICS OF LOSSY-COUNT ALGORITHM ROMEO AND JULIET ENGLISH (K=14456)

Metric	Value
Execution Time (s)	0.00349
Theoretical Max Overestimation Error ($\frac{n}{k}$)	1.0
Final Delta Δ	1
Actual Max Overestimation Error	0
Theoretical >= Actual Max Overestimation Error	True
Mean Absolute Error Top 5	0.0
Mean Relative Error Top 5	0.0%
Mean Absolute Error Top 10	0.0
Mean Relative Error Top 10	0.0%
Mean Absolute Error Top 15	0.0
Mean Relative Error Top 15	0.0%
Mean Absolute Error Top 20	0.0
Mean Relative Error Top 20	0.0%
Mean Absolute Error Top 25	0.0
Mean Relative Error Top 25	0.0%
Mean Absolute Error Top 30	0.0
Mean Relative Error Top 30	0.0%

C.8 Memory Usage Analysis - k = 14456

The memory usage as each word is processed highlighted on Table XXVII and Figure 8 is exactly like the respective Table IV and Figure 1 for the Exact Count Algorithm for the same book:

- The estimated word counts match the exact word counts from the Exact Counter Algorithm, with mean absolute and relative errors of 0.0 and 0.0%, respectively;
- The memory occupied by the word count dictionary also exhibits the same evolution/pattern, ending up occupying 383016 bytes;
- The number of buckets of the word count dictionary also exhibits the same evolution/ pattern, ending up with a maximum number of buckets of 3734, which corresponds to the number of distinct words in the book;
- The pattern showcased by Figure 8 is exactly the same as the one from Figure 1 of the Exact Count Algorithm, with the same order of magnitude on both axis.

One curious detail to mention related to the Lossy-Count Algorithm with $k = 14456$ is that the delta value only evolves on the last processed word, which theoretically makes perfect sense.

Lastly, it's important to mention that as the value of k increases and the algorithm starts behaving more and more like the Exact Counter Algorithm the execution times decrease: 0.00434, 0.00402, 0.00373 and 0.00349 seconds for $k = 100$, $k = 500$, $k = 3000$ and $k = 14456$, respectively. This arises from the overhead present in the Lossy-Count Algorithm due to computing new values of delta and performing pruning operations, which become less and lesser common as the value of k increases.

TABLE XXVII
MEMORY USAGE BY WORD STATISTICS OF LOSSY-COUNT ROMEO AND JULIET ENGLISH (K=14456)

Word	isNewWord	Word Count	Delta	Memory Usage	#Buckets
			0	64	0
tragedy	True	1	0	320	1
romeo	True	1	0	376	2
juliet	True	1	0	432	3
william	True	1	0	488	4
shakespeare	True	1	0	552	5
contents	True	1	0	744	6
prologue	True	1	0	808	7
act	True	1	0	864	8
scene	True	1	0	920	9
public	True	1	0	976	10
place	True	1	0	1312	11
scene	False	2	0	1344	11
ii	True	1	0	1400	12
street	True	1	0	1456	13
scene	False	3	0	1456	13
iii	True	1	0	1512	14
room	True	1	0	1568	15
capulet's	True	1	0	1664	16
house	True	1	0	1720	17
...	...	...	...	...	...
scene	False	6	0	1976	21
open	True	1	0	2568	22
place	False	2	0	2568	22
...	...	...	...	...	...
story	False	2	0	383016	3734
woe	False	13	0	383016	3734
juliet	False	178	0	383016	3734
romeo	False	298	0	383016	3734
exeunt	False	34	1	383016	3734

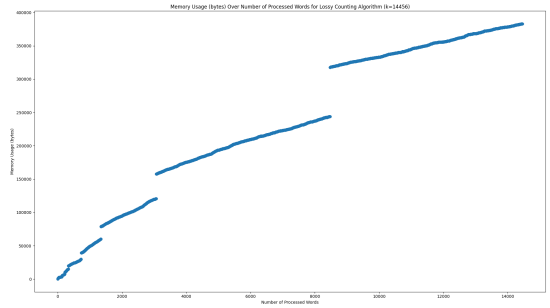


Fig. 8 - Memory Usage (B) by Number of Processed Words for Lossy Count Algorithm — Romeo and Juliet English (k=14456)

V. Fixed Probability Algorithm VS Lossy-Count Algorithm

In the previous sections, the Fixed Probability Algorithm (with $p = \frac{1}{2}$), as well as the Lossy-Count Algorithm were analyzed using the Exact Counter Algorithm as the baseline for comparison. In this section, a final comparison overview of the performance between the Fixed Probability Algorithm and the Lossy-Count Algorithm will be presented.

The Fixed Probability Algorithm presented very accurate results, when multiple (10,000) iterations were performed to obtain the word count estimates. However, this comes at a cost of compute power and memory usage, which defeats the purpose of utilizing probability to decrease the memory consumption of the Exact Counter Algorithm if the multiple iterations are all performed on the same machine. For a single iteration, the algorithm displayed an higher variability and volatility derived from its probabilistic nature. However, even then the results were satisfactory although not as exact as for the 10,000 iterations (which is to be expected). Therefore, depending on the degree of accuracy or error tolerated (for instance for $p = \frac{1}{2}$ the algorithm can never make an exact estimate on the word count of a word that appears an odd number of times) for the intended outcome this algorithm may also be a valid approach that only occupies half the space of the Exact Counter alternative. Although the values of number of iterations and probabilities presented in the report, other values of p and number of iterations would need to be investigated regarding each use case and tolerated error.

However, the reduction in required memory usage of the Fixed Probability Algorithm pales in comparison with the one obtained with the following data-stream algorithm: Lossy-Count Algorithm. On the one hand, although the Fixed Probability Algorithm with $p = \frac{1}{2}$ has allowed to essentially save roughly half the memory in comparison with the Exact Count Algorithm, it still exhibits a linear growth pattern that is not bounded nor limited. On the other hand, throughout this report, it's been shown how by varying the inverse of the error parameter k ($k = \frac{1}{\epsilon}$) it's possible to tune the algorithm to correctly identify the top k of words, while bounding the memory consumption according to the delta Δ value. While the value of k can be in fact tuned to accommodate a predefined top x of words, it shouldn't be increased excessively as the algorithm will start behaving like the Exact Count Algorithm, with a higher memory consumption and a higher number of buckets. As a result of all this factors, the Lossy-Count Algorithm presents itself as the most superior algorithm, when the goal is to obtain the top x most frequent words (and if overestimates are acceptable). Despite this, if the goal is to obtain the list of the least frequent words this is harder to achieve with the Lossy-Count algorithm as it will keep a limited amount of buckets and overestimate them by delta Δ. In this case, the Exact Count Algorithm or the Fixed Probability Algorithm may be better approaches.

VI. Conclusion

This report presented a comparative analysis of three algorithms designed to address the "heavy hitters" problem (and also identify the least common elements): the Exact Count Algorithm, the Fixed Probability Algorithm with $p = \frac{1}{2}$ and the Lossy-Count Algorithm (data-stream algorithm). An analysis of the computational efficiency and limitations of the developed approaches was carried out throughout the report with focusing points being: the absolute and relative errors of the estimates (using the Exact Counter Algorithm as the baseline), validation of whether the same most and least frequent words are identified and ranked in the same relative order, as well as memory consumption and execution time (for the different algorithms and book languages).

VII. Disclaimer

All of the experiments mentioned in these report were done for all Romeo and Juliet books for english, french and german languages. The analysis and discussion for the book in all languages was not covered in this report so not to increase the size of the report any further and because the drawn conclusions would be similar. However all of the results were delivered and are included in the project result's files.

References

[1] GeeksforGeeks, "Lossy counting algorithm - system design", geeksforgeeks.org. Accessed: Dec 27, 2024. [Online.] Available: https://www.geeksforgeeks.org/lossy-counting-algorithm-system-design/.

[2] Project Gutenberg, "Project gutenberg", gutenberg.org. Accessed: Dec 27, 2024. [Online.] Available: https://www.gutenberg.org/.

[3] Project Gutenberg, "Romeo and juliet by william shakespeare", gutenberg.org. Accessed: Dec 27, 2024. [Online.] Available: https://www.gutenberg.org/ebooks/1513.

[4] Project Gutenberg, "Romeo und julia by william shakespeare", gutenberg.org. Accessed: Dec 27, 2024. [Online.] Available: https://www.gutenberg.org/ebooks/6996.

[5] Project Gutenberg, "Roméo et juliette by william shakespeare", gutenberg.org. Accessed: Dec 27, 2024. [Online.] Available: https://www.gutenberg.org/ebooks/18143.

[6] Python Software Foundation, "string — common string operations", docs.python.org. Accessed: Dec 27, 2024. [Online.] Available: https://docs.python.org/3/library/string.htmlstring.punctuation.

[7] PyPI, "nltk - natural language toolkit", pypi.org. Accessed: Dec 27, 2024. [Online.] Available: https://pypi.org/project/nltk/.

[8] Graham Cormode and Marios Hadjieleftheriou, "Finding frequent items in data streams", *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1530–1541, Aug. 2008.
URL: `https://doi.org/10.14778/1454159.1454225`

[9] Python Software Foundation, "collections — container datatypes", docs.python.org. Accessed: Dec 27, 2024. [Online.] Available: https://docs.python.org/3/library/collections.htmlcollections.Counter.

[10] Pympler Project, "asizeof - memory size analysis for python objects", pympler.readthedocs.io. Accessed: Dec 27, 2024. [Online.] Available: https://pympler.readthedocs.io/en/stable/library/asizeof.html.

[11] Kozanakyel, "The core of python dict built-in functions: Understanding hashing and collision resolution", medium.com. Accessed: Dec 29, 2024. [Online.] Available: https://medium.com/@kozanakyel/the-core-of-python-dict-built-in-functions-understanding-hashing-and-collision-resolution-f546c7072fa8.