IT UNIVERSITY OF DENMARK

# Some algorithmic title

Andreas Precht Poulsen Mark Thorhauge Mikkel Hvilshøj Funch

December 17, 2014

# 1 Introduction

## 1.1 Git

The code is available on GitHub on the following link `https://github.com/markthor/AlgorithmDesign`

## 1.2 Project aim

The project aims to discover interesting facts about actors, movies, etc. using data provided by IMDb[1], using algorithms appropriate specific space and time requirements. rewrite

## 1.3 Problem statement

- The first problem is finding the most represented roles in a movie database, using a minimum amount of space. The representation of a role is defined as the percentage of times the role occurs among all roles of all movies.
- The second problem that the project aims to solve, is finding the Jaccard similarity of movie pairs, above a certain similarity threshold, using a minimum amount of time. The Jaccard similarity is based on a predefined subset of the movie attributes.

## 1.4 Algorithmic solution

- The solution to the first problem will be based on the Misra Gries streaming algorithm. The problem is interpreted as a stream by viewing each genre of a single movie as a stream object.
- The solution to the second problem is calculated using locality sensitive hashing with minhashing of shingles relevant to movie data.

## 1.5 Problem scope

For the first problem, the space consumption is measured by the auxiliary space that the algorithm use. The data stream will consume some space, which is not a part of this measurement.

## 1.6 Problem setting

The data to be analysed is a dataset produced by IMDb. The data set is constructed to be loaded into a database and contains a lot of information about movies, actors, etc. However it is not all data which is relevant for the problem, so the dataset have been cleaned for all uneccesary data and a new datafile only containing roles from movies is created and used.

[1] Internet Movie Database, `www.imdb.com`

## 2 Finding the most frequently occurring genres

In the following section, the problem is discussed, as well as different approaches to solving it.

what problem? the problem discussed in ...

### 2.1 Terminology and notation

A role is a string that represents an actor's role in a movie. The data stream, denoted *S*, contains roles, with some of them being identical. The set of most occurring roles are called heavy hitters, denoted *H* and are defined by $\alpha$, the fraction of *S*, that a role constitutes to be in *H*. Let the number of roles in *S* be denoted *m*, and the number of occurrences of a role in *S* denoted *c* then the role is in *H* if and only if $\alpha \leq \frac{c}{m}$. The threshold of the reservoir sampling algorithm is denoted *t*.

### 2.2 Finding heavy hitters

Different algorithms solves the problem of finding the heavy hitters *H* in a data set *S*. A naïve approach is to store all distinct roles in the data set, and their respective count. A space concerned approach to finding the heavy hitters, as described in the section describing the Misra Gries algorithm, can reduce the space consumption.

Reservoir sampling as well

### 2.3 Naïve solution

The naïve solution stores all distinct roles in order to find the roles with a frequency above the threshold. The algorithm contains a map with every distinct role as key, and their respective count as the corresponding value. The time complexity is O(m) while the worst case space consumptions is O(m).

Ninh was /care with time complexity

### 2.4 Reservoir Sampling

The reservoir sampling solution attempts to find heavy hitters from a stream but it does not guarantee it. The set of elements returned by the algorithm, *R*, is in *H*, such that $R \subseteq H$, but not necessarily $H \subseteq R$. Therefore the algorithm might report false negatives. The algorithm is still relevant due to its time complexity and space consumption. The time complexity is $O(n)$, and the space consumption is $O(\frac{t}{\alpha})$. The threshold *t* is the minimum amount of times a given role, must be present in the reservoir in order to be reported as included in *H*.

As with the naïve solution, the algorithm only require a single pass through initial the dataset. After the first pass through, the algorithm counts the frequency of each item in the reservoir and reports the element if the frequency is equal to or exceeds the given threshold. Furthermore, it requires considerably less space than the naïve solution at the expense of guaranteeing that the result is correct. It does so by having a single reservoir array with size $O(\frac{t}{\alpha})$, defined as mentioned above. At first the array is filled with the first $O(\frac{t}{\alpha})$ elements in the dataset, hereafter all new elements have a chance, equal to $\frac{t/\alpha}{n'}$ where *n'* is the number of elements that have been processed at the given point in time, of being swapped with a random element already in the the array.

## 2.5 Misra Gries

The Misra Gries algorithm finds heavy hitters in a data stream. It maintains a map with up to $k$ entries, with keys being data element identifiers. If the set of element keys in the map is denoted $K$, then $H \subseteq K$, when the algorithm has finished. The size of the map is $k = \frac{1}{\alpha} + 1$, meaning that the fraction an element has to constitute of $S$ to be in $H$ has an inverse relation to $k$. For each element in the data stream, the map is updated. If the key of the element already exists in the map, the value is incremented by 1. Else if the size of the map is less than $k$, then the element is added to the map, with a value of 1. If the element does not exist in the map and the size of the map is $k$, then the value of each key is decremented by 1 and keys with value zero is removed. This decrement procedure is repeated until the size of the map is less than $k$ such that the element can be added to the map.

### 2.5.1 Worst case example

The soundness of the algorithm can be explained by constructing a worst case scenario. If $H \subseteq K$, then an element occurring $\alpha$ times must be included in the final map. Consider a stream, with an element $e$ being in $H$ with $\alpha = 0.2$. The stream consists of 10 elements. The map has 6 entries in this example. $e$ occurs twice as the first two, and the most effective way of reducing the count of $e$ is by filling the map and presenting an element that is not yet included in the map. Because it takes 5 distinct elements to fill the map and one additional element to reduce the count of all elements, 6 elements are required to reduce $e$. The stream has 10 elements and therefore this can only happen once. Therefore the count of $e$ is at least 1, when the algorithm terminates.

### 2.5.2 Additional filtering

As Misra Gries returns false positives, additional filtering is needed for K=H. This is done by running the naïve solution, however the counting map only counts elements in K, which makes it as space efficient as Misra Gries.

Den skal gerne rettes til at være single pass. False positives og false negatives er tilladte. Der skal beskrives sandsynligeheder for korrektheden

### 2.5.3 Space consumption

The auxiliary space used by Misra Gries is $O\left(\frac{1}{\alpha}\right)$, but as $\alpha$ is a constant in many applications, this is regarded as constant space consumption.

Hvornår er alpha ikke en konstant

### 2.5.4 String alignment

The data source contains roles of different formats. To make roles that seem similar be regarded as the same string and remove uninteresting roles, the data is cleaned before being parsed to the algorithm. This involves removing strings tagged with symbols and strings that is not regarded as the kind of roles that we wish to investigate.

# 3 Finding similar movies

## 3.1 Jaccard similarity

Movie similarity is defined as Jaccard similarity of sets of a subset of movie attributes. Given two sets $A$ and $B$ the Jaccard similarity coefficient, $j$, is defined as $j = \frac{|A \cap B|}{|A \cup B|}$. The attributes considered when measuring similarity is title, genres, actors and directors. A set of actors is the identifiers of the actors having roles in the movie, a set of directors is the identifiers of the directors, and a set of genres is the names of the genres of the movie. The set that movie similarity is measured from is the union of all these sets as well as the title of the movie and is denoted $C$.

## 3.2 Min Hashing

.... The purpose of Min Hashing is reducing the size the comparable movie representation, such that it still preserves its Jaccard similarity when being compared to other MinHashed objects, as well as making the representation suitable for locality-sensitive hashing.

Indsæt når mark er daun

Min Hashing transforms $C$ to a signature, $S$, which contains an integer for each hashing function used. The signature is generated by applying $n$ hash functions $h_1(C), h_2(C), \dots, h_n(C)$ to the set, which produces a vector $[h_1(C), h_2(C), \dots, h_n(C)]$ of size $n$, each element being the result of a hash function.

The hash functions are defined using permutations. A permutation is a random ordering of all possible values of elements of $C$, which is the union of all the sets that are to be compared. From a permutation, the hash value is computed by iterating through each element of the permutation, in the random order of the permutation. When an element is found that exists in $C$, the hash function returns the index of that element in the ordering of the permutation.

The reasoning for using Min Hash function to calculate signatures is that given two sets, $C_1, C_2$, the probability of $h(C_1) = h(C_2)$ is equal to the Jaccard similarity coefficient.

$$P(h(C_1) = h(C_2)) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

While Min Hashing produces a signature consuming significantly less space than $C$, it stochastically preserves similarity.

For the representation to be suitable for locality-sensitive hashing, each hash function has to be applied on each $C_1, C_2, \dots, C_m$, producing $m$ vectors $S_1, S_2, \dots, S_m$, resulting in a matrix that is the input for the locality-sensitive hashing.

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $h_1(C)$ | 714 | 55 | 10034 | 183 |
| $h_2(C)$ | 2 | 5213 | 3921 | 5213 |
| $h_3(C)$ | 8322 | 377 | 475 | 15632 |

Table 3.1: In this example, the 714'th element of the permutation of the universal set for $h_1(S)$ is the first element that the universal set and $S_1$ had in common, starting from element 0.

# 4 Implementation

## 4.1 String alignment

The data containing the roles and the string describing the actual role contains superfluous information, e.g. the id of the actor, the id of the movie . The string describing the role can for instance contain information defining what specific policeman it was. In order to generalise the roles such information is removed, e.g. there is no difference between policeman #1 and policeman #2 in the cleaned dataset. Additionally unspecified roles are deleted. As an example multiple people has had a role playing themselves. Such a role is defined as "Themselves" in the dataset, which is very general and not the actual role, hence these are deleted. Other similar roles, e.g. himself, herself, extra or additional are removed as well. Some roles define which year the role was from. This information is also removed. Finally, many of the string describing the roles are empty. The entries are deleted.

## 4.2 Data processing

There are multiple ways to handle the data that needs to be processed. One approach is to load all the data into the RAM. This technique is very demanding on the RAM, since everything is stored. Storing everything in the RAM has some clear advantages e.g. being able to go back and forth in the data or reading the same subset of the data multiple times. However if the size of the data exceeds the amount of RAM available, one might experience severe performance implications, due to e.g. memory swapping. Some datasets have a size which makes it impractical, sometimes even impossible, to store them in the RAM. This constraint leads to another approach, which is to stream the data. Streaming the data has advantages as well as disadvantages. Only a fraction of the whole data is stored in the RAM, imposing no extraordinary demand to the size of the available RAM. Only storing part of the data makes it cumbersome, and to some extent impossible, to go back and forth in the data.

## 4.3 Practical space consumption

The following logarithmic graph is the number of map entries used by Misra Gries, reservoir sampling and the naive solution based on the concrete data set. The vertical axis is space consumption and the horizontal axis is $\alpha$. Only the naïve solution is dependent on the size of the concrete example and not on $\alpha$. The reservoir sampling algorithm is linear dependent on the threshold.

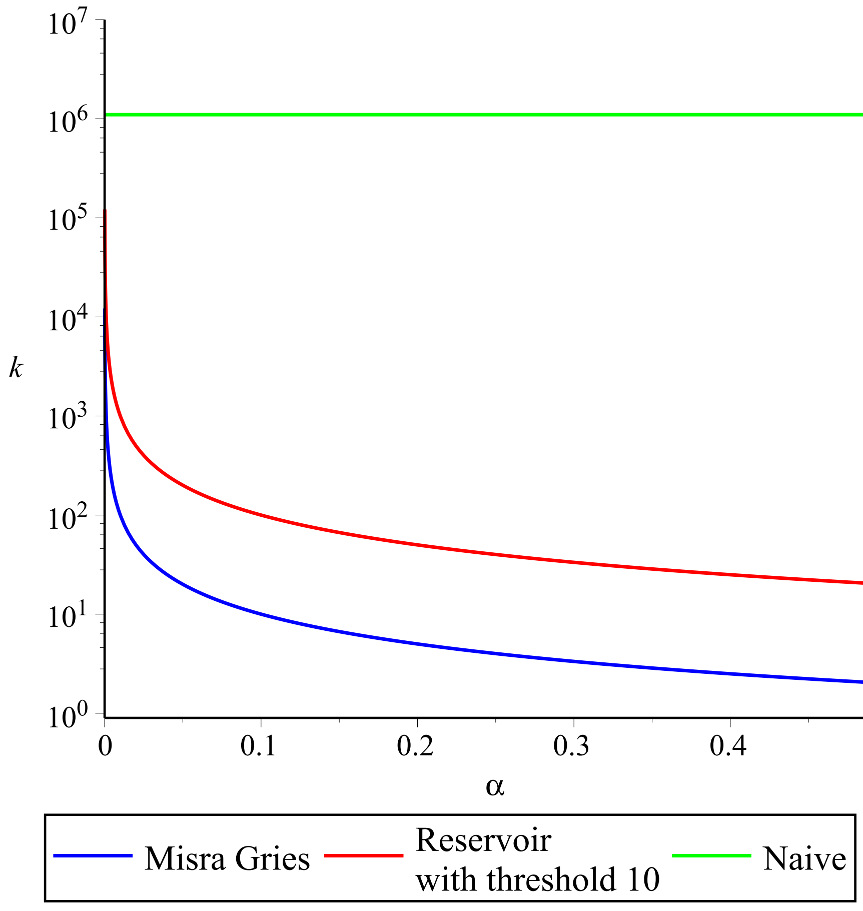

Figure 4.1: Space consumption

As seen in figure 4.1 the space used by the naïve solution is 1000 times as large as the space used by Misra Gries and reservoir sampling. when looking for elements in Hwith =0.001. The size of the data stream is approximately 2.3 million.

$$\begin{aligned}
Pr[a_\ell \text{ is the final } s] &= Pr[a_\ell \text{ is chosen as } s \cdot \prod_{i=\ell+1}^{m} Pr[a_i \text{ does not replace } a_\ell \text{ as } s] \\
&= \frac{1}{\ell} \cdot \prod_{i=\ell+1}^{m} \left(1 - \frac{1}{i}\right) \\
&= \frac{1}{\ell} \cdot \prod_{i=\ell+1}^{m} \frac{1-i}{i} \\
&= \frac{1}{m}
\end{aligned} \tag{4.1}$$

$$\begin{aligned}
Pr[a_{\ell_1}, a_{\ell_2} \text{ form the final sample}] &= \frac{2}{\ell_2} \cdot \prod_{\ell_1 < i < \ell_2} \left(1 - \frac{2}{i} + \frac{2}{i} \cdot \frac{1}{2}\right) \cdot \frac{2}{\ell_2} \cdot \frac{1}{2} \cdot \prod_{\ell_2 < j \le m} \left(1 - \frac{2}{j}\right) \\
&= \frac{2}{\ell_1 \cdot \ell_2} \cdot \prod_{\ell_1 < i < \ell_2} \frac{i-1}{i} \cdot \prod_{\ell_2 < j \le m} \frac{j-2}{j} \\
&= \frac{2}{(m-1) \cdot m} \\
&= 1 \Big/ \binom{m}{2}
\end{aligned} \tag{4.2}$$

## 5 Heading on level 1 (section)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus:

$$\begin{aligned}(x+y)^3 &= (x+y)^2(x+y)\\ &= (x^2+2xy+y^2)(x+y)\\ &= (x^3+2x^2y+xy^2)+(x^2y+2xy^2+y^3)\\ &= x^3+3x^2y+3xy^2+y^3\end{aligned} \tag{5.1}$$

Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies

### 5.1 Heading on level 2 (subsection)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

$$A = \begin{bmatrix} A_{11} & A_{21} \\ A_{21} & A_{22} \end{bmatrix} \tag{5.2}$$

Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem.

#### 5.1.1 Heading on level 3 (subsubsection)

Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim.

**Heading on level 4 (paragraph)** Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim.

## 6 Lists

### 6.1 Example for list (3*itemize)

- First item in a list

    - First item in a list
        - First item in a list
        - Second item in a list
    - Second item in a list
- Second item in a list

### 6.2 Example for list (enumerate)

1. First item in a list
2. Second item in a list
3. Third item in a list