University of Bergen

Masters thesis of Informatics:Algorithms

Practical implementation of a parametrized binary matrix clustering algorithm

Student:
Oskar L. F. Leirvåg
Mentor:
Petr A. Golovach

November 22, 2021

Acknowledgements

There are no words that can express my appreciation to all those who has supported me during this thesis, and all the challenges the world has faced. I'd love to list every singe one by name, but I'm afraid I'd never finish if I did. You all know I have never been good with names, so please assume ignorance, not malice. And know, that no matter how long since we last spoke, I like to think my friends are for life, so please give me a call.

In no particular order;

To all my friends and family, thank you for being my advisors, my fellow procrastinators, my personal trainers, my live entertainers, and my private tutors.

To my old favorite teachers, you're the reason I could even start my degree. You did so much for so many, and you might not even remember, but you saw me for who I was, not just another student. Jorn Hafver, you taught me once, how to find an interest in absolutely anything, and you told me all the secrets on how to manage and customize my degree before I even knew what I wanted to do. Roy Olav Høynes, you once "threw" me out of class, 2 weeks before the exam. You said with a smile that I was "demotivating others" by "answering all the questions", so you gave me a list of extra work, and told me not to be back before the exam. From that day, I knew what I wanted to do.

To all my collages in TF, and especially my team, so many times have you given me advice applicable to my thesis without any idea that you did. Simply by working with you, you kept giving me ideas, even before I knew I needed them. You have, what can only be described as an aura, of knowledge and information around you guys.

To my previous colleagues in SBanken, you're a major reason I wanted to work for a higher degree in the first place, and so many of you advised me to continue studying, even with the temptation of a full-time position. If you read this, you're allowed to say it, yes you did tell me so. No, you may not re-roll that stealth-check; now everyone, roll initiative.

To my unintentional "mini supervisor"; Martin Vatshelle. I do not know where you find the time, or how you manage to cover all the work you do, and still manage to tutor me in other courses. Somehow you always seem to have the time to help anyone and everyone, your students or not, at work or late sundays. You can't possibly know, just how indispensable your help has been, and how much your time is appreciated.

To my supervisor Petr A. Golovach; Even though supervision is an expected part of the job, there are simply some of you that make it seem like it's your entire job. You have been the single stable entity, the point of origin, keeping me on track, even when the entire world froze. You have always went out of your way, and have demonstrated a patience I would not think humanly possible. You once asked me if I wanted a task to complete, or a task from which I would learn. I'm glad I did choose the latter, because you helped me achieve, what I thought I could not.

Lastly, to my better half, life-manager, and girlfriend, Ingrid Johansen; You never seem to fail in making my days better. You have not shown me anything but utter support, and believed in me when I felt I could not. I can't imagine doing this without you.

Abstract

In the [Fomin et al., 2020b] paper, the authors gave an exact parameterized algorithm for the Binary r-Means clustering problem, parameterized by $k+r$, with the runtime of $2^{\mathcal{O}(\sqrt{rklog(k+r)logr})} * nm$. The problem of Binary r-Means takes as input

an $n \times m$ binary matrix $\mathbf{A}$ with columns $(\mathbf{a}^1, ..., \mathbf{a}^n)$, a positive integer $\mathbf{r}$ and a nonnegative integer $\mathbf{k}$. The task is then to decide whether there is a positive integer $\mathbf{r'} \leq \mathbf{r}$, a partition $\{I_1, ..., I_{r'}\}$ of $\{1, ..., n\}$ and vectors $(\mathbf{c}^1, ..., \mathbf{c}^{r'}) \in \{0,1\}^m$ such that $\sum_{i=1}^{r'} \sum_{j \in I_i} d_H(c^i, a^j) \leq k$, where d_H is the Hamming distance.

As the Binary r-Means problem is NP-complete, we cannot expect an algorithm in polynomial time. Therefore its performance as a practical implementation should be evaluated manually in detail, before we make any conclusions about it's viability. An implementation of the algorithm was therefore developed and tested against randomly generated binary matrices, as to measure its performance on a standard desk-top computer.

The results show that even a sub-optimal implementation is inconceivably better than a brute force, and viable with small parameters.

Contents

List of Algorithms

1 Introduction

In the current age of technology, humans have gathered unfathomable amounts of data wherever we saw the possibility to log something. Ranging from major corporations to single individuals, as long as we do anything digital, data is being logged, and the amount can only be estimated to be up in Exabytes. It's well known that there is useful information not only in the direct statistics of this data, but in the anomalies, patterns and correlations between the data. The process of looking though and comparing all this data, has become known popularly as Data mining, due to the complexity of the work, and time required.

One technique of many to find correlations between data in big datasets is clustering. While clustering, we in general look on data-points and organize the data-points into sets; where any data-point within a set, is more similar to all other data-points inside the same set, than all other data-points in other sets. Hence by adding any new data-point, we should be able to find which cluster or classification-set it is closest too, thereby also classifying it. [Gan et al., 2007]

Now suppose we have a streaming webpage that collects data on whether a user like any given movie or not, then we might want to use this data to recommend movies to any specific user which they might like. If we can classify our users into groups that share mostly the same taste in movies, then the "average-user" or centres of the classifications would be a "super user" most likely to share the same movie interest as any other user in the same classification.

To quickly group the new users, we can use these averages or centres. This is called centroid-based clustering, so that all data-points can be linearly classified by finding it's closest centre. The problem then becomes defining what is a centre, and how do we find them.

Unfortunately we now know that to cluster and create these centres accurately is an NP-hard problem, and to create such centres for small instances might not be viable even with massive computing power, over long time frames.

1.1 Motivation

Much research has gone into developing theoretical algorithms for the most complex computational problems in computer science, however many of them remain purely theoretical, not because they're not useful, but because the implementation of brute-force algorithms is simpler, and computational power can be bought. Testing the viability and practicality of implementing these complex algorithms is an important step in the research done for improving the algorithms, and the many practical limitations we face while constructing them in our preferred languages.

This paper aims to clarify the issues faced during development, and the results one could expect on a practical level of computation time on a smaller home computer/laptop.

1.2 Problem statement

From example above with our streaming webpage, we get a rather unique situation where the problem consist purely of binary data-points. We want to find a viable centroid-based clustering algorithm, to find centres in clusters of binary vectors, in a reasonable runtime.

For this paper, we will look into a theoretical centroid-based binary vector clustering algorithm, proposed by [Fomin et al., 2020b]. First we need to find a distance metric between the vectors so we can group them. Because we're clustering binary vectors/matrices, the standard clustering metric is the Hamming distance ($d_H(a, b)$) between the binary vectors. We will here represent the input of these vectors as a single matrix, where each column is a binary vector.

This proposed theoretical algorithm is a parameterized algorithm, limiting the runtime of the algorithm parameterized by the two parameters k and r, where k is the total maximum deviancy of all data-points to their respective centres, and r is the maximum allowed clusters and thereby total centres allowed in a solution.

Now lets look back to our example. We want to find the "centres" or "super users" which we can use to give new movie recommendations to new users. Let's see how we have to encode this problem towards a more formal language.

To convert this problem to a binary matrix, we can define each individual user as a vector $u = \{0, 1\}^m$ where the individual bit-number in the vector represent the same movie for all users

and indicates whether the user liked ('1') or disliked the movie ('0'), such that if the two user-vectors $u = \{1, 1, \ldots\}^m$ and $v = \{0, 1, \ldots\}^m$ we say that u and v both like movie 2, but disagree on movie 1 as $u_1 \neq v_1$.

We then transform the vectors and place them in the set U^n. Now we can use the user-vectors of U as columns in a matrix such that

$$A(U) \rightarrow \begin{pmatrix} 1 & 0 & \cdots & a_{1,n} \\ 1 & 1 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

When running the algorithm we also have to select the parameters k and r where r would be how many centres of movie-personalities we wish to define, and k how much the users in the training-set would be allowed to deviate from the centre in total, also known as the cost.

The resulting vectors would now be the new set of super-users to use in a recommendation system, linearly comparing every new user to the one centre they are closest to whenever they rate movies.

Now let's formally define the problem then, of "Binary r-Means".

Binary r-Means

Input: An $n \times m$ binary matrix $\mathbf{A}$ with columns $(\mathbf{a}^1, ..., \mathbf{a}^n)$, a positive integer $\mathbf{r}$ and a nonnegative integer $\mathbf{k}$

Task: Decide whether there is a positive integer $\mathbf{r'} \leq \mathbf{r}$, a partition $\{I_1, ..., I_{r'}\}$ of indices $\{1, ..., n\}$ and vectors $(\mathbf{c}^1, ..., \mathbf{c}^{r'}) \in \{0, 1\}^m$ such that

$$\sum_{i=1}^{r'} \sum_{j \in I_i} d_H(c^i, a^j) \leq k$$

Now given any set S, it is trivial to find the centre of that set in polynomial time. The challenge here is that we don't know the sets, and we need a centre to create the sets. It simply becomes a problem of which comes first, the chicken or the egg.

Because we need to know the sets, before we can find the centres in the algorithm, we will generally need to compare all data-points to all other data-points, creating a $\mathcal{O}(n!)$ runtime. This is highly undesirable as only a few thousand vectors could result in a runtime longer than the lifetime of the universe. The algorithm from the paper [Fomin et al., 2020b] has however proposed a runtime of $2^{\mathcal{O}(\sqrt{rklog(k+r)logr})} * nm$; placing the algorithm in the FPT class, which is much closer to the runtime we might expect from normal computers.

1.3 Related work

The Binary r-means problem was believed to be NP-complete and was proven to be so for $r \geq 2$ in [Feige, 2014]. Research for exist PTAS (Polynomial time approximation schemes) for this, and for a similar, well known problem called k-means clustering, as in [Kumar et al., 2010]. The resent paper [Fomin et al., 2020a], provides for instance, a randomized linear time approximation scheme for clustering binary vectors.

In the paper [Fomin et al., 2020b, Theorem 4], they show that Binary r-means is FPT when parameterized only by k, but yields a runtime of $2^{\mathcal{O}(klogk)} * nm$, with other related problems.

Thus, excluding brute-force, it does not currently appear to be any exact algorithm for Binary r-Means except those in [Fomin et al., 2020b].

1.4 Overview

The thesis is organized as follows. In section 2 on page 8 the most basic notations and definitions which will be used in later sections are stated. Then section 3 on page 10 introduces the algorithms

implementation in three main parts, and explains how they work in some detail. Section 4 on page 15 presents all the tools used in this paper, and states which custom objects were defined and used in the actual implementation. After that section 5 on page 17 details how the testing was designed and run, how the data was stored, processed and analysed, before detailing the results. Finally in section 6 on page 23 we have the conclusion, and a suggestion for further work. The entire work and results is based on the code available at [Leirvåg, 2021a].

2 Preliminaries

2.1 Notations and Definitions

2.1.1 Matrices

As we are clustering binary vectors $v \in \{0,1\}^m$, we can transform any set of vectors V into a matrix $A(v^1, v^2, \cdots, v^n)$ where n is the number of columns and m is the length of the vectors.

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

2.1.2 Hamming distance

For clustering we need to compare the columns of matrices by distance, to calculate the distance between columns we will use the hamming distance. Let us recall that the hamming distance between two binary vectors $x, y \in \{0,1\}^m$ where $x = (x_1, ..., x_m)^T$ and $y = (y_1, ..., y_m)^T$, is

$$d_H(x, y) = \sum_{i=1}^{m} |x_i - y_i|$$

or, in other words, the number of positions $i \in \{1, ..., m\}$ where x_i and y_i differ.

With that distance we can also define the distance between two matrices as the sum of the distance between the columns of the matrices. Now we can also define the cardinality of a binary vector as the number of positive bits in the vector $v \in \{0,1\}^m$ such that $d_C(v) = \sum_{i=1}^{m} v^i$.

$$d_H(A, B) = \sum_{i=1}^{n} d_H(a^i, b^i)$$

2.2 Majority rule

To find a single optimal binary centre $s \in \{0,1\}$ in a set of binary columns I, it must follow, that for each row, the bit in s_i must be the same value as the most prevalent bit in the row I_i.

2.3 Parameterized algorithms

As the main algorithm in this thesis is based on a parameterized problem, we here refer to the book [Cygan et al., 2016] for a detailed introduction. Here we shall only introduce the most basic notions.

Definition 1 (Parameterized problem)**.** *[Cygan et al., 2016, p. 12] A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.*

With the problem of r-means from [Fomin et al., 2020b], we here consider it as such a parameterized problem, with $(A, k + r)$

Definition 2 (Fixed-parameter tractable)**.** *[Cygan et al., 2016, p. 13] A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called fixed-parameter tractable (FPT) if there exists an algorithm $\mathcal{A}$ (called a fixed-parameter algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) * |(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT*

By this definition the proposed algorithm from [Fomin et al., 2020b] should have a runtime in the form of $f(k) * |(x, k)|^c$. We can see this in the proposed runtime which is proposed to be $2^{\mathcal{O}(\sqrt{rklog(k+r)logr})} * nm$.

Definition 3 (Data reduction rule)**.** *[Cygan et al., 2016, p. 18] A data reduction rule, of a parameterized problem Q is a function $\phi\Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance (I, k) of Q to an equivalent instance (I', k') of Q such that ϕ is computable in polynomial time in $|I|$ and k. We say that two instances of Q are equivalent if the following holds: $(I, k) \in Q$ if and only if $(I', k') \in Q$.*

Definition 4 (Kernelization, kernel)**.** *[Cygan et al., 2016, p. 18] A kernelization algorithm, or simply a kernel, for a parameterized problem Q is an algorithm $\mathcal{A}$ that, given an instance (I, k) of Q, works in polynomial time and returns an equivalent instance (I', k') of Q. Moreover, we require that $size_{\mathcal{A}}(k) \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.*

Definition 5 (Turing kernelization)**.** *[Cygan et al., 2016, p. 314] Let Q be a parameterized problem and let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. A Turing kernelization for Q of size f is an algorithm that decides whether a given instance $(x, k) \in \Sigma^* \times \mathbb{N}$ is contained in Q in time polynomial in $|x| + k$, when given access to an oracle that decides membership in Q for any instance (x', k') with $|x'|, k' \leq f(k)$ in a single step.*

3 Algorithm

3.1 Kernelization

The goal of the preprocessing algorithm is to reduce the input matrix by following certain reduction steps based on the two parameters k and r, running in polynomial time over the input size, here $(n \times m)$.

Designing the algorithm it should require an $n \times m$ binary matrix A. And the two parameters $r > 0$ limiting the maximum number of clusters, and $k \geq 0$ limiting the cost of all total distances. By definition 4, it must produce an equivalent problem. However, the way we will construct this algorithm, it instead outputs several small matrices $L = A_1, \ldots, A_s$, which can be clustered separately, but whose separate solutions produce a combined solution if and only if the original matrix A has such a solution with the same r and k. Hence acting like a turing kernel as in definition 5.

The first observation is that if there are more than $k + r$ distinct columns, there can not be a solution, so we can simply return NO. Therefore as a start we partition the columns into clusters of equal columns $I = I_1, \ldots, I_t$ and test if $t \geq k + r$. We will call this partitioning I and its groups for the *initial clusters*. Further we can assume that all columns off initial clusters will be clustered together, so if the size of such an initial cluster $|I_i|$ is larger than or equal to $k + 1$, we assume that there must be a centre with zero distance to this cluster. Thus, any initial cluster of size $|I| \geq k+2$ can be reduced down to $|I| = k + 1$.

Now, we can greedily group the columns of the initial clusters together, with any other columns of distance $d_H(I_a, I_b) \leq k$ and call this new partition L. From the way we constructed these sets of columns, we now have that the distance between any two columns from any to separate sets, is of a distance $d_H > k$, and cannot be clustered together within the cost of k.

Instead of treating this as a single solution, we can process each of the sets individually as its own matrix, henceforth a *sub-matrix*. Now we can remove what we can call *uniform rows*. A uniform row, is a row consisting only of zeros or only ones. If we remove a uniform row, it will not have any impact on the final result as the optimal centre should keep the uniformity of the row for zero cost. Since all columns were clustered together with those of distance $d_h \leq k$, the number of pairwise distant columns is at most $k + r$, and we can thus conclude that the number of non-uniform rows is bound.

All this is in fact explained by the following lemma

Lemma 1. *[Fomin et al., 2020b, Lemma 5] Given an instance (A, r, k) of Binary r-Means, Algorithm 1 runs in $O(n^2m)$ time and either correctly concludes that (A, r, k) is a no-instance of Binary r-Means and returns NO or returns $s \leq r$ matrices $A_1, \ldots, A_s$ such that*

- *(A, r, k) is a yes-instance of Binary r-Means if and only if there are positive $r_1, \ldots, r_s$ and nonnegative $k_1, \ldots, k_s$ such that (i) $r_1 + \cdots + r_s \leq r$, (ii) $k_1 + \cdots + k_s \leq k$, and (iii) (A_i, r_i, k_i) is a yes-instance of Binary r-Means for every $i \in \{1, \ldots, s\}$*
- *for every $i \in \{1, \ldots, s\}$, A_i is $m_i \times n_i$ matrix with $m_i \leq max\{(\ell_i - 1)k, 1\}$, where ℓ_i is the number of distinct columns of A_i and $n_1 + \cdots + n_s \leq (k + 1)(k + r)$*
- *the total number of distinct columns in $A_1, \ldots, A_s$ is at most $k + r$*

Algorithm 1 Pre-processing for Binary r-Means

Input: An $n \times m$ binary matrix $A = (a^1, ..., a^n)$, $r > 0 \wedge k \geq 0$
Output: A set L of sub-matrices $A_1, \ldots, A_s$

procedure PREPROCESS(A, r, k)
  $I \leftarrow \{1, ..., n\}$
  Construct the partition $\{I_1, ..., I_t\}$ of I into initial clusters
  if $t > k + r$ **then**
    Return NO
  else
    for $i \leftarrow l$ to t **do** ▷ Reduce duplicates down to $k + 1$
      while $|I_i| > k + 1$ **do**
        Select arbitrary $j \in I_i$
        $I_i \leftarrow I_i \backslash \{j\}$ and $I \leftarrow I \backslash \{j\}$
      end while
    end for
    $s \leftarrow 0$
    while $I \neq \emptyset$ **do**
      $s \leftarrow s + 1$
      Select arbitrary $j \in I$
      $S \leftarrow \{j\}$ and $I \leftarrow I \backslash \{j\}$
      while $\exists((p \in S) \cap (q \in I))$ *s.t.* $d_H(a^p, a^q) \leq k$ **do** ▷ Produce clusters greedily
        $S \leftarrow S \cup \{q\}$
        $I \leftarrow I \backslash \{q\}$
      end while
      $J \leftarrow \{1, ..., m\}$
      while $|J| \geq 2$ and $A[J, S]$ has uniform row with an index $h \in J$ **do**
        $J \leftarrow J \backslash \{h\}$ ▷ Remove uniform rows if $|J| \geq 2$
      end while
      $L \leftarrow L \cup A[S, J]$
    end while
  end if
  if $s > r$ **then**
    Return NO
  else
    Return L
  end if
end procedure

From lemma 1 we know that a combination of the solutions of the individual sub-matrices in L should produce a solution for the initial binary matrix A if and only if A has such a solution, thus we actually have a definition 5. The papers original pseudo-code [Fomin et al., 2020b] gives an algorithm to create a substitute matrix "gluing" together these clusters into one single matrix, but as they can be processed individually, we will do that instead.

3.2 Branching Algorithm

Now from the kernel in algorithm 1 we are given a list L of sub-matrices, which can be processed individually. Here we shall look at the main step of the algorithm; the Binary r-Means, recursive branching algorithm.

By the recursive nature, the input should consist of a set of indices $I \subseteq \{1, 2, \ldots, n\}$ referencing un-clustered columns from A_i, and a set S with the already selected centres. We also need the parameters k and r. Initially $S := \emptyset$, and $I := \{1, 2, \ldots, n\}$. We also expect to see a change in the parameter k, so we will rename it to d as to represent the current available cost.

For each recursion, we must check weather the branch has either produced a solution, or exceeded the parameters allowed. We know that we have a set of selected centres S, so if S can

cover all remaining columns within the cost d we have the solution and return S. If however we have $|S| = r$ then we have a NO-instance and return.

If the algorithm can continue, we can conclude that the optimal solution given the set S must have at least one more element. Thus we try to find centres of minimum distance less than d to another column. We shall here call the currently selected minimum distance h. We start from zero, and work h up until d. Now any remaining columns, whose distance to any centre in S, is less then h, will be closer to a column currently in S than a new centre, and can therefore be covered by S, removed from I, while adjusting the remaining budget d appropriately.

Now, to find the new centre we can either branch or brute force. The efficiency of which is determined by the formula

$$|I| \leq \sqrt{dr \ log(m) \ / \ log(r)}$$

If we can brute force, we try all partitions of I with size p up to either $|I|$ or the remaining budget of r, whichever comes first. Then for all clusters j in a partition J_j we find the optimal means s^j for the cluster, by using the *majority rule*. And test if it produces a solution when adding $\{s^1, \ldots, s^p\}$. If so, return S

If however we cannot brute force, then we must branch. We must try all possible vectors $s \in \{0,1\}^m$ whose distance to a remaining column in I is equal to h. We iterate over all the vectors s, and recurse by calling r-Means with $S = S \cup \{s\}$. If we find a solution then return it.

If no solution is found, we must increase the minimum cost h. Though, if h is no longer less than d there cannot be any solution, and we return NO.

Algorithm 2 Binary r-Means

Input: A set of indices $I \subseteq \{1, \ldots, n\}$, a set of vectors $S \subseteq \{0,1\}^m$, and a nonnegative integer d
Output: A set S of centres

procedure R-MEANS(I, S, r, d)
  if $\sum_{i \in I} min\{d_H(s, a^i) | s \in S)\} \leq d$ **then**
    Return C = S
  end if
  if $|S| = r$ **then**
    Return NO
  end if
  $h \leftarrow 0$
  while $h \leq d$ **do**
    foreach $i \in I$ **do**
      if $l = min\{d_H(s, a^i) | s \in S\} \leq h - 1$ **then**
        $I \leftarrow I \backslash \{i\}$ and $d \leftarrow d - l$
      end if
    end for
    if $d \geq 0 \wedge |I| \leq \sqrt{dr\ log(m)\ /\ log(r)}$ **then**
      for $p := 1$ **to** $min\{|I|, r - |S|\}$ **do**
        foreach *partition* $\{J_0, \ldots, J_p\}$ *of* I, *where* J_0 *may be empty* **do**
          for $j := 1$ **to** p **do**
            find the optimal means s^j for the cluster J_j using the majority rule
          end for
        end for
        $S \leftarrow S \cup \{s^1, \ldots, s^p\}$
        if $\sum_{i \in I} min\{d_H(s, a^i) | s \in S)\} \leq d$ **then**
          Return C = S
        end if
      end for
    end if
    if $h \leq d/|I|$ **then**
      foreach *vector* $s \in \{0,1\}^m$ *s.t.* $d_H(s, a^i) = h$ *for some* $i \in I$ **do**
        Call r-Means$\{I, S \cup \{s\}, r, d\}$
        if *the algorithm returns a solution* C **then**
          Return C
        end if
      end for
    end if
    $h \leftarrow h + 1$
  end while
  Return NO
end procedure

With a correct implementation this algorithm should produce and return a new set S containing the new proposed centres for each cluster. The sum of distances from any column in A to the closest center in S should then be less than or equal to d, and the number of centres in S should be less than or equal to r.

3.3 Dynamic programming

The original r-means kernelization algorithm by [Fomin et al., 2020b] intended for the sub-matrices $L = A_1, \ldots, A_s$ produced from the kernel in section 3.1 on page 10 to be *"glued"* together into a single matrix before running the r-means algorithm. However as of lemma 1 then instead of stitching the solution into a single matrix A' we can run the algorithm multiple times over A_i with increasing parameter k, save the different possible solutions in a set H *(hypotheses)*, and stitch

the solutions in the end. Thus saving memory space and allowing the sub-matrices to be run in parallel.

For every $i \in \{1, \ldots, s\}$ and $0 \leq k' \leq k$, we define

$$f_i(k') = \begin{cases} \text{min \#clusters that can be obtained for} A_i \text{ using at most budget } k' \\ r+1, \qquad \text{if \#clusters} > r \end{cases}$$

We have in section 3.2 on page 11 given the algorithm for finding the min #clusters that can be obtained for A_i, using at most budget k'. Thus $f_i(k')$ makes use of Binary r-Means() with $0 \leq d \leq k'$, and $1 \leq r' \leq r$.

Using the tables of values from $f_i(k')$, we define the table $T[i, k']$, where the elements are defined as follows

$$T[i, k'] \rightarrow \text{min \#clusters that can be obtained for the set of sub-matrices } \{A_1, \ldots, A_i\}, \text{ using at most budget } k'$$

Which we can define with the functions $f_i(k')$

$$\begin{aligned} T[1, k'] &= f_1(k') \\ T[i, k'] &= min\{min\{T[i-1, k_1] + f_i(k_2) \mid k_1 + k_2 = k' \text{ and } k_1 \wedge k_2 \geq 0\}, r+1\} \end{aligned} \quad (1)$$

As soon as the table $T[s, k]$ is complete, then (A, k, r) is a YES instance if $T[s, k] \leq r$

Lemma 2. *The recurrence in* (1) *correctly computes the values of* $T[i, k']$

Proof. (Lemma 2) For $i = 1$ (1) holds by the definition of $T[1, k']$ and $f_1(k')$

Let then $i \geq 2$ and assume that (1) holds for all lesser values of i. We show (1) by proving two inequalities. First we show that

$T[i, k'] \geq min\{min\{T[i-1, k_1] + f_i(k_2) \mid k_1 + k_2 = k' \text{ and } k_1 \wedge k_2 \geq 0\}, r+1\}$

then that

$T[i, k'] \leq min\{min\{T[i-1, k_1] + f_i(k_2) \mid k_1 + k_2 = k' \text{ and } k_1 \wedge k_2 \geq 0\}, r+1\}$

For all i, let $l_i = \{A_1, ..., A_i\}$ be the set of sub-matrices from 1 to i

---

$T[i, k'] \geq min\{\ldots\}$

Let r_1 be a solution to l_{i-1}, r_2 be a solution to A_i, k_1 be the cost of r_1, and k_2 be the cost of r_2.Thus we have $r_1 \geq T[i-1, k_1]$ and $r_2 \geq f_i(k_2)$ and thereby $r' = T[i, k'] = r_1 + r_2 \geq T[i-1, k_1] + f_i(k_2) | k_1 + k_2 = k'$

$T[i, k'] \leq min\{\ldots\}$

Let k_1 be the cost of a solution r_1 for l_{i-1}, and k_2 be the cost of a solution to A_i. Then $r_1 = T[i-1, k_1]$ and $r_2 = f_i(k_2)$, from which we can conclude that $T[i-1, k_1] + f_i(k_2) = r_1 + r_2 \geq T[i, k'] | k_1 + k_2 = k'$

□

3.4 Summary

Now lets summarize the complete algorithm.

As first we call the pre-processing algorithm (kernel) in section 3.1 on page 10 on the problem to hopefully reduce the input into smaller inputs L.

Then given these instances $L = \{A_1, \ldots, A_i\}$ we can find the minimum solution for any sub-matrix by calling the **Binary r-Means()** algorithm from section 3.2 on page 11, from within the final dynamic algorithm in section 3.3 on the previous page. The dynamic programming algorithm allows us to store each result, avoid calling the hard method unnecessarily many times, also producing the result for the original initial binary matrix A.

From the following theorem we also have the complete algorithms runtime.

Theorem 1. *[Fomin et al., 2020b, Theorem 5]*
Binary r-Means is solvable in $2^{\mathcal{O}(\sqrt{rklog(k+r)logr})} * nm$ *time.*

4 Algorithm implementation

4.1 Tools used

4.1.1 The Java programming language, Version 17

Selecting the programming language for any project is always an issue, and the choice of using the Java programming language was mostly based on availability and the existing operations and packages in the language, rather than the runtime.
The Java programming language is not as fast as many of its alternatives, but does run almost seamlessly on any operating system due to running on the JVM (Java virtual machine).

4.1.2 GCP / BigQuery

Running several thousand of tests multiple times and storing all results to a database with input and resulting output can easily fill large amounts of data. To avoid a bottleneck the testing computer all results could be pushed to an external database. For this the cloud database "BigQuery" in the "Google Cloud Platform" could be used.

4.1.3 Google Data studio

The advantage of using an external database solution like BigQuery is that it's also easily integrated in other tools. "Google Data studio" can create statistical plots based on large data systems, and is able to directly query the BigQuery database for almost live updates.

4.1.4 JetBrains - IntelliJ IDEA

For any development in a high-level programming language an intelligent development environment (IDE) is usually recommended. The IntelliJ IDEA is a premium alternative of many IDEs.

4.1.5 Git / GitHub

During development of an application it's recommended to use a version-control tool. GIT is one of the most known version-control-tools in the field, and allows the user to traverse the entire solutions history at any time.

To use GIT as a tool, an external GIT hosting solution can improve the experience while acting as a backup tool. While a simple server could be sufficient, advanced solutions like GitHub exist to add extra functionality.

4.1.6 Maven

Maven is a package manager for the Java Programming language, allowing the user to easily list the needed dependencies of their project, then handling the rest.

4.1.7 Travis

Travis is a build tool that can seamlessly integrate with most Git solutions such as GitHub to automatically fetch the newest version of any repository, and run build scripts defined within the project. It also allows to run unit-tests and push the report.

4.1.8 Codacy

Codacy is a code quality evaluation tool, that can seamlessly integrate with most Git solutions such as GitHub to automatically fetch the newest version of any repository, and run a quality control on the code to catch any typical errors, bad habits, or simply code that breaks standards to enhance readability and further development.

Codacy also holds a test-report functionality, though it lacks the ability to compile and run any code or tests, it allows other tools to push the test-results in such that in can serve as a total quality-control dashboard for the project.

4.1.9 Combinatoricslib3

Combinatoricslib3 is a open source Java library available on GitHub with existing combinatorial functions such as generating permutations.

4.1.10 Testing computers

The algorithm tests were run on a two testing computers

- "Lenovo YOGA 900" with "Intel Core i5 (6. gen) 6200U / 2.3 GHz" 8GB (1600 MHz / PC3-12800), LPDDR3 SDRAM. 256GB SSD M.2 memory, Intel HD Graphics 520. Running the minimal operating system, Linux distribution ArchLinux.
- A custom built desktop with "AMD FX-8350", "16.0GB Dual-Channel DDR3 @ 722MHz" memory, and a 4095MB NVIDIA GeForce GTX 980 graphics card. Running the more standardized operating system "Microsoft Windows 10".

Also see section 6.1 on page 23 for more information.

4.2 Data structures

4.2.1 BinaryVector

The first logical implementation would be to simply use a *Boolean array*, which would most likely work, but due to the implementation of the Boolean type in the Java Programming language, it still uses 1byte *1 Byte = 8 bits*, which is usually not a problem, but if we duplicate a 1000×1000 bit array $1'000$ times, the size becomes *1 GigaByte*. If we can create a smarter solution using bits, the same matrices could be stored in one eight of the space *125 MegaBytes*

A solution for this is to use the bits directly, like using numbers to store the bits and use the standard bit-operators of any programming language. Unfortunately this creates a limit in the length of the vectors *long = 64 bit* and their mutability which would require workarounds, but could theoretically improve both the runtime, and the memory.

Luckily, the Java Programming language, standard library has an existing type whit a similar implementation. Each binary vector is represented as a BitSet, which is not human-readable, but a BitSet could be presented as a set of positions which is set to 1. Hence the binary byte representation of the number 42 could be represented as the set $42 \equiv 00101010 \equiv (3, 5, 7)$. The Java library version of BitSet also includes several useful binary operations and extras such as cardinality. It should though be noted that the BitSet standard is not Thread-safe [Oracle, 2021]

4.2.2 BinaryMatrix

The main Binary matrix structure is defined as a list of vectors or BitSets, along with the original height and width.

4.2.3 BinarySubMatrix

The binary sub matrix is mainly copies of a binary matrix, and along also stores the parent binary matrix reference. This allows for deletion of rows and columns, for deleted rows and columns to be restored, and to find original positions of all rows and columns in the parent BinaryMatrix.

4.2.4 BMAHypothesis

The BinaryMatrixApproximationHypothesis objects, were created to structure the multiple results produced iterating over all possible parameters. It is only a container for a proposed solution to a BinarySubMatrix with a given parameter d, holding the used cost d and the centres produced.

4.2.5 Partition

The partition object, was defined to extend on the Java standard library *ArrayList<List<T>>* structure and allow the creation of partitions from a list of objects.

5 Testing

5.1 Generating test matrix

For the generation of a matrix used for testing we could use any complete randomized binary matrix, but to be able to control the result after the completion of the algorithm it is best to create more concentrated tests.

By reversing the entire algorithm we can generate a set of *initial vectors* H of length h, which will be the centres, and by picking random vectors from the set H into a new set S, we will create a new random binary matrix A, with a solution of $k = 0$, and $r = h$.

By then flipping random bits, we can slightly modify the expected result, mainly the k parameter. By flipping d bits in A, we will still have a solvable matrix with $r = h$ as long as $k \geq d$ though the matrix might still be solvable within both smaller k and r.

Algorithm 3 Test-Matrix generation

Input: Two nonnegative integers d and h, and two positive integers n and m
Output: A binary matrix A of size $n \times m$, from h *initial clusters*, with d randomly flipped bits

```
1:  procedure GenerateTestMatrix(d, h, n, m)
2:      while h ≥ 0 do                          ▷ Generate random initial columns
3:          H_h ← RandomVector(m)
4:          h := h − 1
5:      end while
6:      i ← 0
7:      while i < n do
8:          S_i := S ∪ H_rnd                    ▷ Add random vector from H
9:          i := i + 1
10:     end while
11:     while d ≥ 0 do
12:         x ← rnd(0, n)
13:         y ← rnd(0, m)
14:         S_xy ← ¬S_xy                        ▷ Flip random bit
15:         d := d − 1
16:     end while
17: return A[S, m]
18: end procedure
```

Still for normal runtime analysis the importance is to test the worst cases for the algorithm, and while the concentrated tests is controllable, most of those will be mostly solved by the kernel, not allowing for proper testing of the actual algorithm. For this the generation algorithm would run with small parameters ($r \leq 3$, and $k \leq 30$).

5.2 Running the test

5.2.1 Setup

The algorithm testing was performed on randomly generated matrices of varying sizes, on the computers specified section 4.1.10 on the previous page, and published at [Leirvåg, 2021b]. Every single test iteration would generate a new matrix, then measure the milliseconds of the single iteration, then push the runtime with metadata to the cloud database specified in section 4.1.2 on page 15.

5.2.2 Countermeasures

The timing was measured directly inside the program, between generation and publishing as to only measure the time used to solve, and discount any time used to generate or latency caused by slow connections. The tests was allowed to run for extended periods, and mostly when the computers were not in use.

It was possible to run multiple tests simultaneously as the both computers is multi-core, making sure not to overload the CPU usage.

5.2.3 Cloud computing

As to avoid the loss of control on the actual hardware running the tests, cloud "mining" was initially avoided. As the testing showed to be very resource intensive however, it was theorized how it could be achieved. Unfortunately many cloud providers have measures in place to avoid misuse in crypto-currency mining, and further discouraged its use.

5.2.4 Data collection

The data for each iteration is its runtime, but metadata was added to separate later for analysis. The metadata added was all the input parameters as $k, r, d, h, width, heigh$, the solution results, whether it was solved, the computer running the test (in case of multiple test machines), and the date-time of the test case.

5.3 Analysis

To analyse the results of the several thousands of tests, a statistical tool was used, specified at section 4.1.3 on page 15. In the tool the result-database was imported directly as a table, and plots were created to visualize the data.

In Data studio, new columns were added to support a multi-parameter plot for better comparison of the algorithms behaviour over the different parameters. This includes combining the variables h and r into a unified h, r variable such to illustrate the different runtimes in the same plots. This same technique was also used to combine the running width and height to compare the runtimes of the matrix-size later.

5.4 Results

First off, let us recall that we defined the variables as such

$h \rightarrow$ *#initial clusters* used to generate the random binary matrices
$d \rightarrow$ *#random-bit-flips* performed on the randomly generated binary matrices
$r \rightarrow$ the initial parameter r that was given to the algorithm
$k \rightarrow$ the initial parameter k that was given to the algorithm

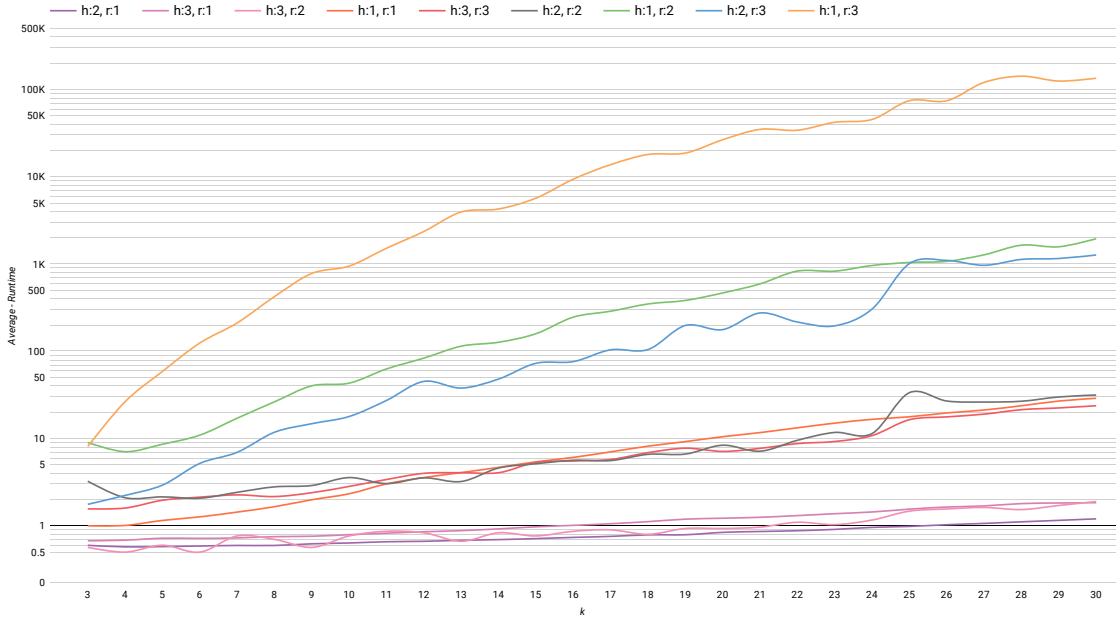


Figure 1: Plot showing the average runtime of the algorithm in milliseconds *(logarithmically)* over the parameter k, using matrices generates of size Width: 1000 and Height:1000.

The first runs was done on several 1000x1000 binary matrices. We can see from the plot in figure 1 that the runtime is a function of both k and r as expected. We can also see that there is a large dependency of on the amount of initial columns generated initially, which is also expected.

Looking at the same matrices over d in figure 2 on the next page, we can assume that the amount of noise introduced to the initial matrices affected the runtimes. The noise variable d is randomly dependent on k but while we see from figure 2 on the following page the average runtime of the worst case $(h : 1, r : 3)$ at $k \simeq 30$ is somewhere around $150'000ms$ or 2 minutes and 30 seconds, the average runtime of the worst case keeps growing exponentially to the worst case $(h : 1, r : 3)$ at $d \simeq 60$ is somewhere around $600'000ms$ or 10 minutes.

The size of the matrix should affect the runtime so the algorithm should be tested on smaller matrices. The following data is from 200x200 matrices. However looking at figure 3 on the next page we see almost the same results as we did in figure 1

In fact almost all matrices seems to follow the same pattern of runtime with little to no effect on the runtime between much larger matrices. This indicates that the kernel is able to mitigate the size efficiently, and that the data-structures used does not severely impact the runtime of the results.

It is then to be expected that the rows are not a large factor of the runtime with a proper implementation, hence the amount of columns and the distance between them has the greatest impact on the runtime, along with the parameters k and r.

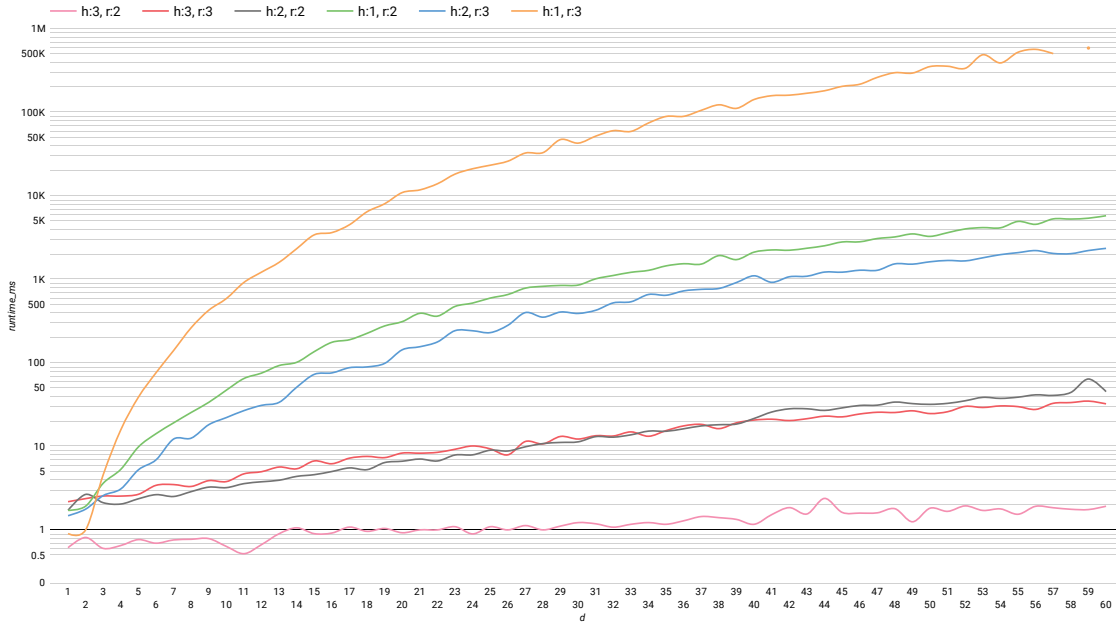


Figure 2: Plot showing the average runtime of the algorithm in milliseconds *(logarithmically)* over the parameter d, using matrices generates of size Width: 1000 and Height:1000.

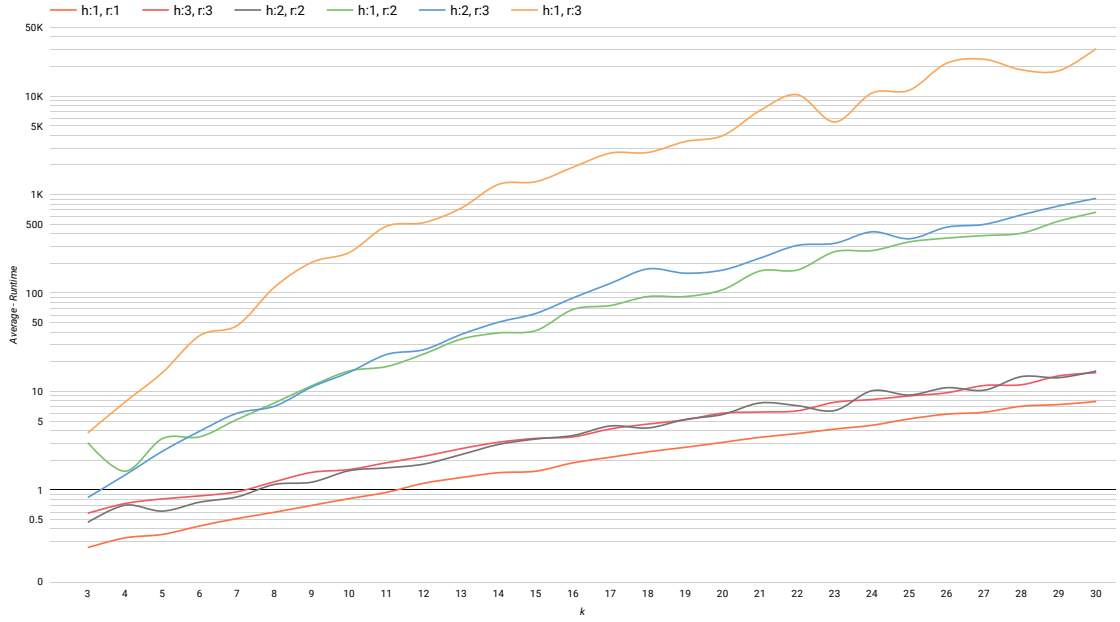
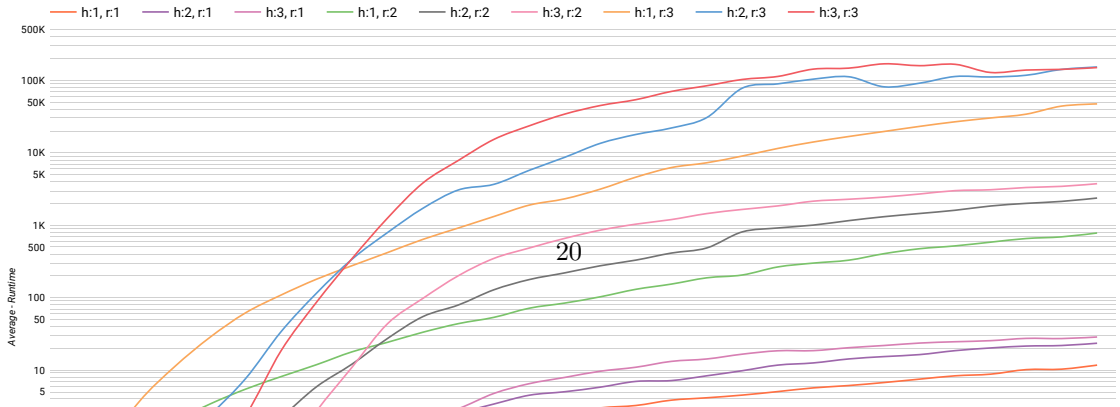


Figure 3: Plot showing the average runtime of the algorithm in milliseconds *(logarithmically)* over the parameter k, using matrices generates of size Width: 200 and Height:200.



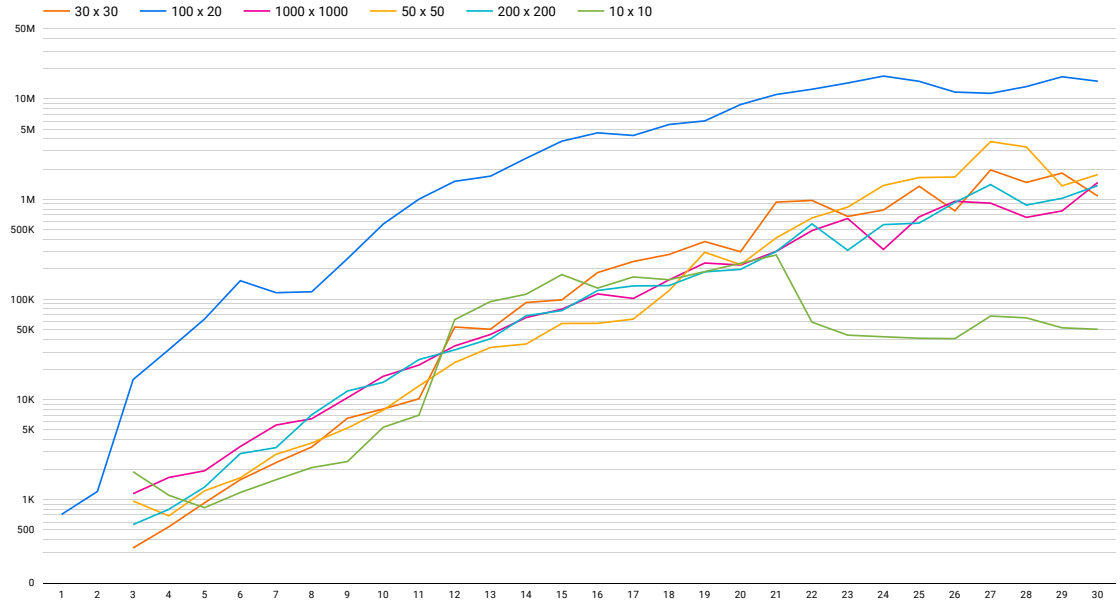


Figure 4: Plot showing the average runtime of the algorithm in milliseconds *(logarithmically)* over the parameter k, all data from all test-runs with all matrices from this report, grouped by the size of the matrices.

Now looking at figure 5 on the previous page we see different results. The runtime behaves quite differently from both the figure 1 on page 19 and the figure 3 on the previous page. Here we can see what appear to be three groups with similar runtime grouped by r. The three groups here are

- $(h1,\ r1), (h2,\ r1), (h3,\ r1)$
- $(h1,\ r2), (h2,\ r2), (h3,\ r2)$
- $(h1,\ r3), (h2,\ r3), (h3,\ r3)$

Compared to figure 3 on the preceding page we can see that it starts off similar, but as k grows, we get a clear change in runtime. The $(h3,\ r3)$ which seemed of no significance, now crosses over $(h1,\ r3)$ with much higher average runtimes consistently. The same goes for $(h2,\ r3)$, which for figure 3 on the previous page was much lower.

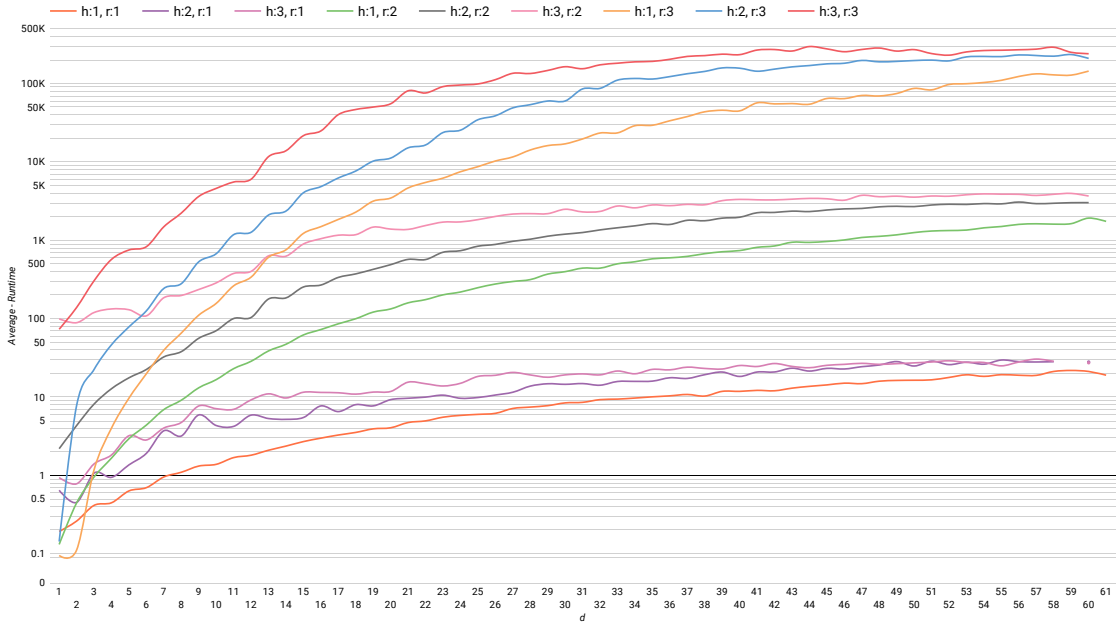


Figure 6: Plot showing the average runtime of the algorithm in milliseconds *(logarithmically)* over the parameter d, using matrices generates of size Width: 100 and Height:20.

One theory is that as k grows, so does the number of noise d, but looking at figure 6, it appears that both $(h3,\ r3)$ and $(h2,\ r3)$ both have a higher avg. runtime than $(h1,\ r3)$.

Looking more at figure 6, it holds the same grouping by r as we saw in figure 5 on page 20, but the runtimes seems more stable from the beginning.

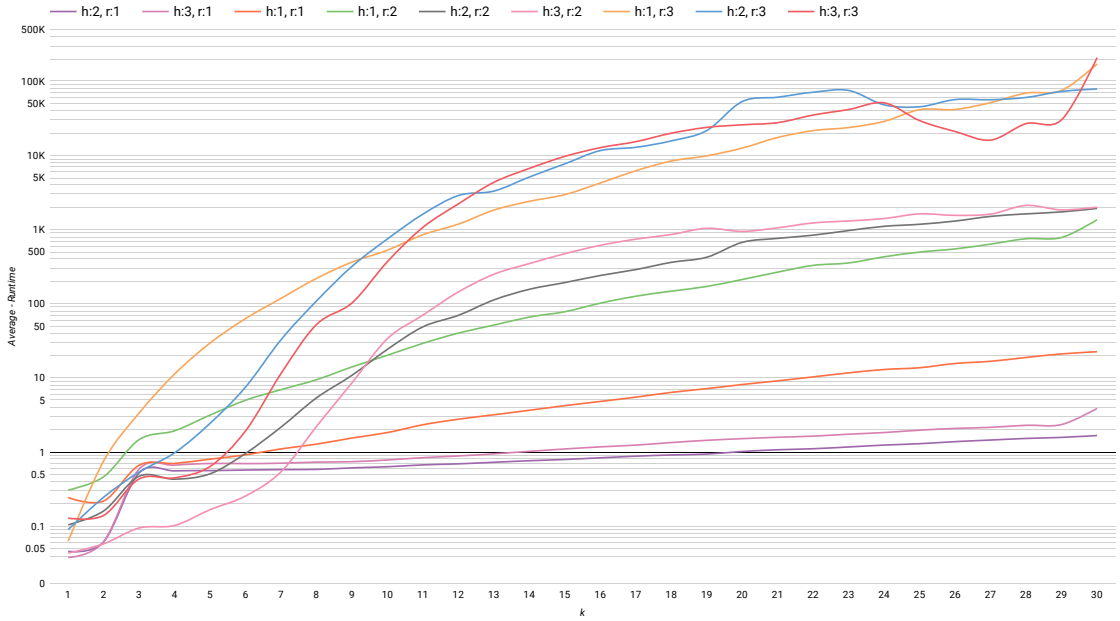


Figure 7: Plot showing the average runtime of the algorithm in milliseconds *(logarithmically)* over the parameter k, using all data from all test-runs with all matrices from this report.

Adding all the data from all matrices of all sizes together, we do se some of the significance of

figure 5 on page 20 bleeding trough, though an non-uniform test-case balance could be the reason for this. What's more important is the overall runtime. Though the matrix size is not a huge factor, we do see that figure 1 on page 19 do impact the result as some of it's worst cases was in the *500Kms* or *8+ minutes*.

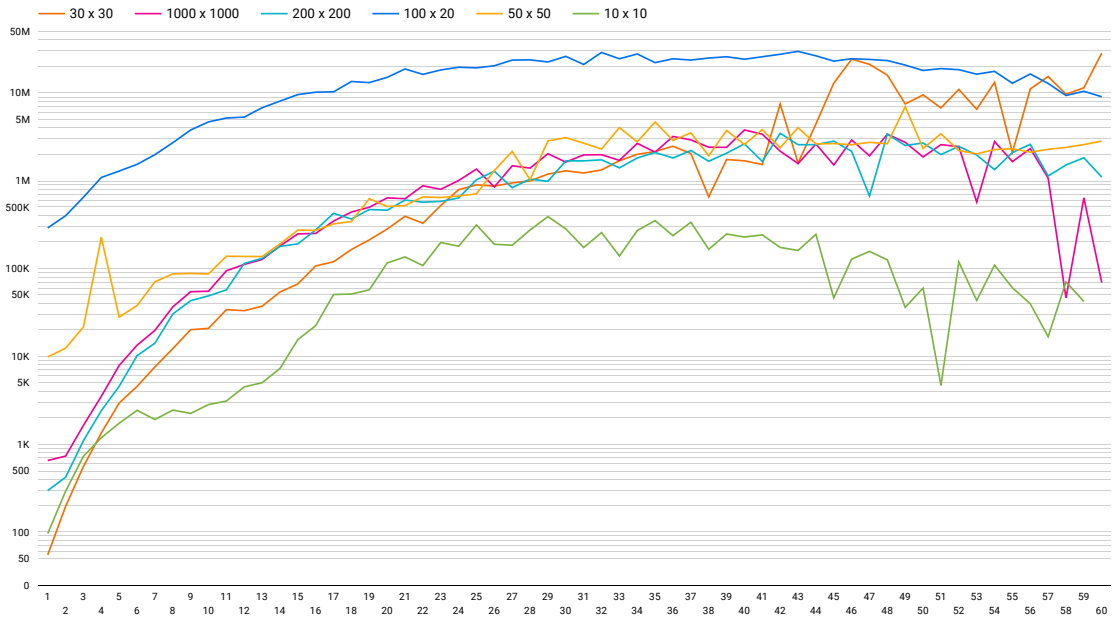


Figure 8: Plot showing the average runtime of the algorithm in milliseconds *(logarithmically)* over the parameter d, using all data from all test-runs with all matrices from this report, grouped by the size of the matrices.

Though the plot appears rather noisy in the larger d-space *(due to the randomness of d)*, seeing all matrices by size over d, we can see the much of the same results as in figure 4 on page 21, and it's clear that on average the 100×20 matrices perform the worst.

6 Conclusion

By the performance seen during testing, it appears as if the algorithm behaves as expected. We saw only a polynomial dependency on the input-matrix, and a much greater dependency on the given parameters r and k.

However, it becomes apparent that this algorithm is in NP, as although inconceivably better than a brute-force implementation, it still has a noticeable runtime, which can stretch from several minutes to hours even for small instances. Thus the question of the algorithms viability still stands.

As for minor implementations of small matrices with little noise, and small parameters, it can be used on smaller devices, and even smart-devices. However, any larger use-cases would wound up in minutes or hours, excluding the every-day use-cases. Instead it could be of use in data-analysis, where larger machines are available, and where time is not limited in seconds.

As this implementation is only a "proof of concept", there are several theoretical improvements that should be tested to improve runtime *see section 6.1*. However, even with the current implementation, the results looks promising.

Still, more experimentation is needed.

6.1 Future work

For stable results, the algorithm though theoretically able to be run in parallel, was only tested as a linear implementation, due to hardware limitations on the testing computer producing highly

irregular results for initial parallel tests. Though the solutions produced were correct, the processing time varied too greatly between several runs, even on the same input. Still with a different implementation it could theoretically be possible to optimize the algorithm further for high parallel processing, maybe even GPU programming.

Most high-level programming languages have a performance drop when using recursion. While recursion is easy to implement, it can usually be avoided with workarounds. If possible, the recursive calls in the main branching algorithm could be implemented in an iterative procedure, possibly improving the runtime significantly.

The programming language was chosen for it's familiarity, and multi-OS practicality, as to produce a proof-of-concept. It is well known, that the Java programming language is sub-optimal in its performance due to several factors. The usage of a language which can get closer to the machine operations, would probably improve runtime significantly.

The testing was done on "typical" home computers, which were both running the tests in the background, not with the possibility of dedicated cores/threads. Though this has probably not affected the results unevenly, it may have affected the overall runtime negatively. A dedicated processing server would eliminate the possibility of operating systems, and random background tasks (updates etc) possibly interfering with the results.

The hardware in the testing computers is of old technology. The average performance is probably no longer what is promised by the manufacturer, and newer technology does on average have better performance. Also the writer was made aware during testing that there existed a dispute on the processor technology in use for section 4.1.10 on page 16; namely the "AMD FX-8350" processor, where it was determined that it is in actuality not an 8-core, but 4-core. [dickey v. AMD, 2015, dickey v. AMD, 2020, Gartenberg, 2019]. This has most likely impacted the accuracy of the runtime somewhat, when more than 4 test were running simultaneously on the affected machine. However isolating the machine shows much of the same trends, but unfortunately holds to few test-cases to be conclusive on it's own.

While the algorithm and it's implementation could theoretically be improved, the dataset used for testing was only random. While working with proper datasets, natural symmetry and other patterns might make the data reducible enough, or open for the possibility of customize the algorithm for a specific data-set.

References

[Cygan et al., 2016] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2016). *Parameterized Algorithms*. Springer.

[dickey v. AMD, 2015] dickey v. AMD (2015). https://angeion-public.s3.amazonaws.com/www.AMDCPUSettlement.com/docs/Complaint.pdf. COMPLAINT AND DEMAND FOR JURY TRIAL, CLASS ACTION.

[dickey v. AMD, 2020] dickey v. AMD (2020). https://www.govinfo.gov/app/details/USCOURTS-cand-4_15-cv-04922/USCOURTS-cand-4_15-cv-04922-16. STIPULATED FINAL ORDER AND JUDGMENT.

[Feige, 2014] Feige, U. (2014). Np-hardness of hypercube 2-segmentation. *arxiv*.

[Fomin et al., 2020a] Fomin, F., Golovach, P., Lokshtanov, D., Panolan, F., and Saurabh, S. (2020a). Approximation schemes for low-rank binary matrix approximation problems. *ACM transactions on algorithms*, 16(1):1–39.

[Fomin et al., 2020b] Fomin, F. V., Golovach, P. A., and Panolan, F. (2020b). Parameterized low-rank binary matrix approximation. *Data Mining and Knowledge Discovery*, 34(2):478–532.

[Gan et al., 2007] Gan, G., Ma, C., and Wu, J. (2007). *Data Clustering: Theory, Algorithms, and Applications*. SIAM, Philadelphia, ASA, Alexandria, VA.

[Gartenberg, 2019] Gartenberg, C. (2019). Amd to pay out $12.1 million in false advertising class action suit over bulldozer chips. https://www.theverge.com/circuitbreaker/2019/8/28/20837336/amd-12-million-false-advertising-class-action-lawsuit-bulldozer-chips.

[Kumar et al., 2010] Kumar, A., Sabharwal, Y., and Sen, S. (2010). Linear-time approximation schemes for clustering problems in any dimensions. *Journal of the ACM*, 57(2):1–32.

[Leirvåg, 2021a] Leirvåg, O. (2021a). https://github.com/RakNoel/MasterProject.

[Leirvåg, 2021b] Leirvåg, O. L. F. (2021b). Google datastudio results. https://datastudio.google.com/reporting/e9511b18-ed5f-4017-a369-a5833000b68f.

[Oracle, 2021] Oracle (2021). Java platform, standard edition java api reference. https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/BitSet.html. Accessed: 2021-08-11.