# DATA DIVERSIFICATION IN DIFFERENT DOMAINS

**APIVICH HEMACHANDRA**

**A SENIOR PROJECT SUBMITTED IN**
**PARTIAL FULFILMENT**
**OF THE REQUIREMENTS FOR**
**THE DEGREE OF BACHELOR OF SCIENCE (PHYSICS)**
**MAHIDOL UNIVERSITY INTERNATIONAL COLLEGE**
**MAHIDOL UNIVERSITY**
**2020**

**COPYRIGHT OF MAHIDOL UNIVERSITY**

# ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge the fact that this senior project is submitted as a requirement for a Degree in Physics, and yet has very little to do with physics.

With this point aside, I would like to thank all of the Ajarns at MUIC who have taught me many things I should know about physics, computer science, maths, and much more. In particular, I would like to thank Aj Kanat who have helped me through this senior project, and Aj Piti who have been a great advisor for the past four years.

I'd also like to thank all the friends I have made along the way, including all those in 1409 who I spent countless hours working, playing games and procrastinating with (I'd list all of your names here but that would take a bit too much time). It's very sad that I have to spend my last semester stuck in my room rather than in 1409. If any juniors are reading this (which is extremely unlikely), please make sure to put up more memes on the Meme Board.

I would also like to thank Ajarn Sarana Nutanong, head of AIResearch Institute and the dean at Vidyasirimedhi Institute of Science and Technology (VISTEC), who gave me an opportunity to do my summer research internship at VISTEC (after a very long process), and to let me continue the project as part of my thesis. I would also like to thank P' Nat, who have kindly agreed to co-advise my senior project and helped me with the project throughout the past year.

Finally, I'd like to thank my family, even though they are unlikely to be reading my thesis. You guys are always very supportive of whatever I do, and I find that very cool of you guys.

Apivich Hemachandra

DATA DIVERSIFICATION IN DIFFERENT DOMAINS.

APIVICH HEMACHANDRA 5980002 ICPY/B

B.Sc. (PHYSICS)

SENIOR PROJECT ADVISORS : KANAT TANGWONGSAN, PH.D., NAT DILOKTHANAKUL, PH.D.,

ABSTRACT

We tackle the problem of data diversification from multiple angles. We first explore the problem of text diversification and how sampled subsets of sentences from different sampling techniques can affect the machine translation model obtained. We show that we are able to get notable increase in translation quality in some cases of our sampling.

We then focus on data diversification on a wider scope and consider the problem of submodular maximisation, which is one way a diversification problem can be phrased more mathematically. We develop a parallel submodular maximisation algorithm under a cardinality constraint, and demonstrate its performance against existing submodular maximisation algorithms. We are able to show that our algorithm yields accuracies comparable to existing algorithms, while taking orders of magnitude less computation time and fewer function calls.

KEY WORDS : DIVERSIFICATION
NATURAL LANGUAGE PROCESSING
SUBMODULAR MAXIMISATION

74 pages

# CONTENTS

**ACKNOWLEDGEMENTS** iii

**ABSTRACT** iv

**1 Introduction** 1
- 1.1 Diversification . . . 1
- 1.2 Contributions and Outline . . . 2

**2 Sentence Diversification for Machine Translation** 3
- 2.1 Introduction . . . 3
- 2.2 Background Information . . . 3
  - 2.2.1 Machine Translation and The Need of Sentence Diversification . . . 3
  - 2.2.2 Word and Sentence Embeddings . . . 4
  - 2.2.3 Distances Between Sentences . . . 6
  - 2.2.4 Sampling and Diversification Algorithms . . . 9
- 2.3 Experiments . . . 12
  - 2.3.1 Pipeline . . . 14
  - 2.3.2 Diversification Methods Tested . . . 15
  - 2.3.3 Corpora . . . 15
  - 2.3.4 Setup For Machine Translation Model . . . 16
- 2.4 Results and Analysis . . . 18
  - 2.4.1 Performance of Sampled Set in Training Translation Models . . . 18
  - 2.4.2 Analysis of the Sampled Subsets . . . 20
- 2.5 Conclusion . . . 24

**3 An Efficient Parallel Submodular Maximisation Algorithm** 30
- 3.1 Introduction . . . 30
- 3.2 Background Information . . . 30
  - 3.2.1 Submodular Maximisation Problem . . . 30
  - 3.2.2 Applications of Submodular Functions . . . 32
  - 3.2.3 Existing Algorithms . . . 33
- 3.3 Subroutines From Previous Works Used . . . 37
  - 3.3.1 The THRESHOLD-SAMPLING Algorithm . . . 38

3.3.2 The EXHAUSTIVE-MAXIMISATION Algorithm . . . . . 43
3.4 A Practical Parallel Submodular Maximisation Algorithm . . . . . 44
3.4.1 A Problem With Parallel Algorithms Discussed . . . . . 46
3.4.2 The two versions of THRESHOLD-SAMPLING . . . . . 46
3.4.3 Our Proposed Algorithm . . . . . 47
3.4.4 Theoretical Analysis of COOL-SUBMOD . . . . . 47
3.5 Tests of Our Algorithm . . . . . 54
3.5.1 Implementation . . . . . 55
3.5.2 Testing Setup . . . . . 55
3.5.3 Algorithms Benchmarked . . . . . 56
3.5.4 Dataset . . . . . 57
3.6 Results . . . . . 58
3.7 Conclusion . . . . . 58

**4 Conclusion** **64**
4.1 Summary . . . . . 64
4.1.1 Sentence Diversification for Machine Translation . . . . . 64
4.1.2 Submodular Maximisation For Large Datasets Under Cardinality Constraint . . . . . 64
4.2 Future Extensions . . . . . 65

**A Omitted Details From Chapter 3** **67**
A.1 A Faster REDUCED-MEAN Algorithm . . . . . 67

**REFERENCES** **70**

**BIOGRAPHY** **74**

# CHAPTER 1

# INTRODUCTION

## 1.1 Diversification

The problem of diversification appears in a large number of contexts, especially in data mining and machine learning. Often, we end up with more data than we will need, or than we can process. We therefore may need to reduce the number of data points that we have. However, if the dataset were to be reduced, it would be nice if the data points selected are important or are diverse with respect to the other data points selected. Specific examples of the diversification problem include the following:

- Data selection for training machine learning models. In order to train a model, we will need both the input data and the corresponding output data. Often, we will not be able to use all of the data that we have, and so we may need to select a subset of the data which might be the best for our training.
- Recommendation systems. If we know the preference for a user, we will be able to work out the items from our catalogue which the user may be interested in. However, to give a variety for the user, the items that we select should also be diverse enough, and represent the entire collection of items.
- Summarisation of data. We have a large number of data points, but we may want to select a small subset of the data points which gives a good overview of the whole dataset. An example is selecting a few pictures which act as a short review of an entire album of photos.

Data diversification methods can be done on many levels. We may explore a specific instance of when data diversification is needed and see which diversifying method is the most useful for our data. On the other hand, we may also view the diversification problem more generally, as a mathematical optimisation problem which can be solved using some algorithm.

In this project, we will be exploring the diversification from both angles. We will explore a specific case of the diversification problem, by testing the usefulness of data diversification in training a machine translation model. We will also consider the diversification problem in the form of submodular maximisation problem, which is a generalisation for many diversification problems stated earlier, and develop an efficient algorithm to approximate its solution.

## 1.2 Contributions and Outline

In this thesis, we will address the following points.

- We will construct a framework for training set selection for neural machine translation tasks. Using the framework, we will test out different sampling techniques, including changing the similarity metric of sentences, and the algorithm to select appropriate sentences. We then test how well the different sampling techniques perform based on how well we are able to train a machine translation model using training data selected according to the different sampling techniques.
- We will develop an efficient submodular maximisation algorithm under a cardinality constraint. We will analyse the theoretical performance of our algorithm, and demonstrate that it is able to give very good answers on real test cases, while requiring much lower running time and costs compared to existing parallel algorithms.

This thesis will be organised as follows.

- In Chapter 2, we will discuss methods to measure sentence similarities, algorithms for diversification, and present results on the effectiveness of these methods and algorithms towards building a diverse dataset for training machine translation models.
- In Chapter 3, we will focus on the problem of submodular maximisation and develop an efficient and practical parallel algorithm for the submodular maximisation under a cardinality constraint.
- In Chapter 4, we summarise our findings and provide possible extensions to this project.

# CHAPTER 2

# SENTENCE DIVERSIFICATION FOR MACHINE TRANSLATION

## 2.1 Introduction

In this chapter, we describe our attempt for sentence selection for reducing the training set size for machine translation tasks. In Section 2.2, we discuss the related background information. Then, in Section 2.3, we describe the setup for the tests we have done. In Section 2.4, we present the results. We finally present the concluding remarks and future extensions in Section 2.5.

## 2.2 Background Information

In this section, we will discuss background information required for our experiments. We will talk about the task of machine translation, and why sentence diversification may be needed. We will then discuss techniques of word and sentence embeddings, and methods of measuring the similarity between the sentences using these embeddings. We finally mention algorithms which can be used to diversify the samples we have according to the similarity metrics we mentioned.

### 2.2.1 Machine Translation and The Need of Sentence Diversification

An interesting problem in machine learning is the problem of machine translation. Here, people aim to create a model which will take in sentences in one language and output the same sentence translated to another language. The problem of machine translation (and for natural language processing in general) can be challenging, as a good model would be able to understand the semantics of a human language beyond the mere characters or morphemes that make up a word. Many deep learning models have been tested for machine translation tasks, including models which contains Long-Short Term Memory (LSTM) units [HS97, SVL14], or the more recent transformer architectures [VSP+17].

A major aspect of training neural networks for machine translation (and in fact for other machine learning models) is acquiring the initial training data, which has to be accurate according to the domain knowledge. For machine translation, texts in a source language and its corresponding translated sentence in a target language is needed for training. These texts can often be scraped from places with the same passages in multiple languages, or can be translated by human translators.

To train more accurate translation models, larger quantities of texts and their corresponding translations could be used. This can be a challenge however, as parallel corpuses may not be readily available and to translate new texts by hand to generate new corpuses often requires a lot of manpower. There have therefore been different techniques proposed which help reduce the number of sentences required for training.

One such technique is active learning, where the model being trained is able to make queries to an oracle and ask about labels of some unlabelled data. The model decides which data should be queried based on some metric such as its uncertainty on the true label of the data. This technique reduces the number of actual labelled training data that is needed to train a model [Set10]. The downside to this method is that the selected samples are not model-agnostic, meaning samples which arise from active learning with one model may not necessarily be helpful to a different model. Additionally, in the context of machine translation, the idea of back-translation has been proposed, where two translation models are trained simultaneously and each model generates a synthetic corpus for the other model to train on [EOAG18]. Again, this trick is model-dependent.

Ideally, to build a good dataset, we would like to select samples which arises independently of any translation models. In this work, we will be proposing a framework for diversifying a source corpus to select a good subset of sentences to perform training for machine translation tasks.

### 2.2.2 Word and Sentence Embeddings

Handling sentences purely as a set of words in some human language is often not useful for machine learning purposes. People have therefore developed models which aim to convert some words or sentences in one language and embed them onto some coordinate space and instead represent them as a set

of multidimensional vectors, which can then be more easily fed into a neural network, for example.

**Word-Level Embeddings**

To get a word embedding from some sentence $S$, we would first have to break the sentence down into a set of tokens ( $t_1, t_2, \ldots, t_n$ ) where each $t_i$ is a member of some set of vocabulary $\Sigma$. These tokens may be the words that makes up the sentence, but they may also be morphemes that makes up the words. Then, to get the word embeddings, some function $f : \Sigma \to \mathbb{R}^k$ would map each token to form the final vector embeddings ( $f(t_1), f(t_2), \ldots, f(t_n)$ ).

A naïve embedding system would be to assign one word found in a sentence to a one-hot vector. Here, one word would be assigned the vector [ 1 0 0 ... ], another word assigned the vector [ 0 1 0 ... ], and so on. Since we assign a unique vector to each word in our vocabulary, the vectors will have a dimension of $|\Sigma|$. While this is simple to do, it require a high-dimensional vector, and often these vectors encode little information about the words themselves as each word can be seen as orthogonal to each other, hence is seen less related. Other techniques for word embeddings therefore have been developed.

Word2Vec is a method of generating vector representation for words which also encodes linguistic aspects of the words themselves [MCCD13]. Syntactic or semantic relationships between words are embedded in the differences of the vectors. For example, the difference between the vector for `easy` and the vector for `easiest` is similar to the difference between the vector for `heavy` and the vector for `heaviest`, since the pairs represents an adjective and its superlative. Similarly, the difference between the vector for `France` and the vector for `Paris` is similar to the difference between the vector for `Spain` and the vector for `Madrid`, since the pairs represents a country name and its capital city name.

Newer vector embedding systems such as ELMo [PNI+18] and, more recently, BERT [DCLT18] also aims at encoding the contextual information of words into their vectors. In contrast to the Word2Vec model, ELMo and BERT does not have a predefined vector embeddings for each words, but instead will embed the words into vectors according to the surrounding contexts. Therefore, unlike in the Word2Vec model, the same words can be embedded to different vectors according to their contexts. Words with similar

meanings will be closer together than homonyms which mean different things. The difference between ELMo and BERT is with the architectures of the model - ELMo is a bi-directional LSTM model, whereas BERT uses a transformer model. BERT outperforms ELMo in various natural language tasks due to the transformer's ability to better retain long term information compared to an LSTM model.

**Sentence-Level Embeddings**

There has also been attempts at finding methods of embedding entire sentences. In other words, given a sentence $S$, we would like to have a function which returns a $k$-dimensional vector representing $S$. An attempt to do such is the Universal Sentence Encoder [CYK+18]. In this case, models based on either the Transformer architecture (which aim to produce more accurate models at higher computation costs) or the Deep Averaging Network (DAN) architecture (which aim to be faster but give lower accuracy then the Transformer model). The resulting embedding is a single 512-dimensional vector. The pre-trained Transformer model can be used through TensorFlow Hub (which is a package available on Python).

### 2.2.3 Distances Between Sentences

A distance function can be thought of as a way to measure the closeness of two objects. If the distance between two points in physical space is large, then this usually means that the two points are far away from each other. With sentences, we want our distance function to reflect the similarity between the two sentences. In other words, if the distance between two sentences is small, then this should mean that the two sentences are similar to each other, in some aspect.

Sentences alone are often unhelpful to determining their distances. The sentences therefore should be embedded or processed in some way before applying a distance function upon it. Below, we discuss how distances can be determined from different embeddings.

**Distances Using Word Embeddings**

Some sentence distances is calculated using the word embeddings of the sentences. One of the most famous of such is the Word Mover's Distance [KSKW15]. This is a modification to Earth Mover's

Distance, and intuitively, it is minimum distance each token in the first sentence needs to "travel" to the tokens in the second sentence. More mathematically, given two sentences $S_1 = \{t_1, t_2, \ldots, t_n\}$ and $S_2 = \{t'_1, t'_2, \ldots, t'_m\}$, we would like to find

$$\begin{aligned} \min_{T \geq 0} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} T_{ij} \cdot d(v_{t_i}, v_{t'_j}) \\ \text{subject to} \quad & \sum_{j=1}^{m} T_{ij} = c_i && \forall i \in \{1, \ldots, n\} \\ & \sum_{i=1}^{n} T_{ij} = c'_j && \forall j \in \{1, \ldots, n\} \end{aligned}$$

where $v_t$ is the vector embedding of token $t$, $d$ is a distance metric for our embedded tokens, and $c_i$ and $c'_j$ are the amount of flow required in or out from a following token, which corresponds to how many times the particular token or token embedding appears in the first or second sentence. The matrix $T$ here can be thought of as the flow matrix, where $T_{i,j}$ is how much of token $i$ in the first word "travels" to token $j$ of the second word, and so $\sum_{i=1}^{n} \sum_{j=1}^{m} T_{ij} \cdot d(v_{t_i}, v_{t'_j})$ represents the total distance each token in the first sentence needs to travel to transform into the second sentence.

We can see however, that computing this distance involves solving a linear program, which is not cheap to do. The relaxed Word Mover's distance therefore have been proposed as an approximation to the version above. To compute this, we find $T'$ and $T''$ where

$$T'_{ij} = \begin{cases} c_i & \text{if } j = \arg\min_j d(v_{t_i}, v_{t'_j}) \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad T''_{ij} = \begin{cases} c_j & \text{if } i = \arg\min_i d(v_{t_i}, v_{t'_j}) \\ 0 & \text{otherwise} \end{cases}$$

and let the relaxed distance be given by $\max_{T \in \{T', T''\}} \sum_{i=1}^{n} \sum_{j=1}^{m} T_{ij} \cdot d(v_{t_i}, v_{t'_j})$.

While the original paper on Word Mover's distance computes this distance on older Word2Vec embeddings, we suspect that it should also be possible to compute such distances using newer word embedding schemes such as BERT. In our tests, we will be using the relaxed version of Word Mover's distance on embeddings from BERT. Since the same token may not be embedded to the same vector using BERT, we will assume that each embedded vectors are unique and let $c_i = 1$ and $c'_j = 1$ for all $i$ and $j$.

**Distances Using Tokenised Sentences**

Rather than using the actual word embeddings of a sentence, simply the tokens that makes up the sentence can be used as well. The tokenised sentences are cheaper to compute than the actual word embeddings themselves as tokenising a sentence is a preprocessing step necessary before obtaining word embeddings anyway.

Treating the sentences as a set of tokens $S_1 = \{t_1, t_2, \dots, t_n\}$ and $S_2 = \{t'_1, t'_2, \dots, t'_m\}$, we can define the Jaccard distance to be

$$d_{\text{Jaccard}}(S_1, S_2) = 1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}.$$

Jaccard distance measures how many tokens are shared between the two sentences. The distance function is written in a way such that $d(S_1, S_2) = 0$ when $S_1 = S_2$. Note that the range of Jaccard distance is $[0, 1]$.

Another distance function that can be used is the edit distance, or the Levenshtein distance. Both of these terms are often used interchangeably, and in this paper we will refer to this distance as the edit distance. Edit distance measures the number of insertion, removal or substitution operations required to transform one string to another. Edit distance is useful in measuring dissimilarity between two strings of letters (where the operations acts on the characters of the string), however we can also adapt edit distance to work with token insertion, token removal and token substitution as well. The problem of calculating edit distance has been solved for a long time, famously using the Wagner-Fischer algorithm which is a classic example of an algorithm implemented using dynamic programming [WF74]. However, faster algorithms to compute edit distance has also been proposed [Hyy02, AN20].

Given two sentences, the maximum possible number obtainable from edit distance is the maximum length between the two sentences. To bound the distance between a range independent of sentence length, we can use normalised edit distance, which is given by

$$d_{\text{norm-edit}}(S_1, S_2) = \frac{d_{\text{edit}}(S_1, S_2)}{\max\{|S_1|, |S_2|\}}.$$

We can see that the range of $d_{\text{norm-edit}}$ is bounded between $0$ to $1$.

**Distances Using Sentence Embeddings**

Instead of the word embeddings, we can calculate similarities of sentences using the vectors generated from sentence-level embeddings. This is easier to do as embedding at the sentence level outputs a single vector, which can directly be compared against the vector representation of other sentences. While we can use Euclidean distance to measure the difference between the vectors, it is more common to use the angular distance, which for two sentences, is defined as

$$d_{\text{angular}}(S_1, S_2) = \frac{1}{\pi} \arccos\left(\frac{\langle \vec{v}_{S_1}, \vec{v}_{S_2} \rangle}{\|\vec{v}_{S_1}\| \|\vec{v}_{S_2}\|}\right)$$

where $\vec{v}_S$ represents the sentence embedding of sentence $S$, $\langle \cdot, \cdot \rangle$ represents the inner product, and $\| \cdot \|$ represents the $L_2$-norm of a vector. Geometrically, this value represents the angle between the two vectors, where we have also added a normalisation factor so that the range fits between $[0, 1]$.

### 2.2.4 Sampling and Diversification Algorithms

Given a set of sentences $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, we would like to find a subset $\mathcal{T} \subseteq \mathcal{S}$ with $|\mathcal{T}| \leq k$, which is said to be a diverse subset of $\mathcal{S}$ of some maximum cardinality $k$. We can see that there are many ways of describing the "diverseness" of the sample $\mathcal{T}$. Here, we provide some diversification objectives and the algorithms to solve for these objectives, along with a geometric demonstration on mock two-dimensional data in Figure 2.1.

**Maximum Sum of Distances**

The first way is to define diverseness in terms of the difference between the elements within the selected set. A simple method to do this is to maximise the sums of the distances of each pair of points in the sample set, i.e. to find

$$\underset{\mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| = k}{\operatorname{argmax}} \sum_{S_1, S_2 \in \mathcal{T}} d(S_1, S_2).$$

In this section, we will refer to this objective as the MAXSUM objective. Many algorithms for diversification have been presented, as summarised in [ZWQ+16]. However, for larger dataset as we are dealing, it is more

practical to use algorithms in the streaming setting, where all data does not have to be loaded at once. A simple streaming algorithm is proposed by [MSN11], which is as shown in Algorithm 1. The key idea of the algorithm is that for each new element in the stream, it greedily replaces an element in our selected subset if doing so will increase the objective function, and discards it otherwise.

**Algorithm 1** A streaming algorithm to diversify for the MAXSUM objective

**function** DIVERSIFY-MAXSUM($\mathcal{S}, d, k$)
  $\mathcal{T} \leftarrow \emptyset$
  $c \leftarrow 0$ ▷ the sum of distance of items in $\mathcal{T}$
  **for** $S$ in $\mathcal{S}$ arriving in a stream **do**
    **if** $|\mathcal{T}| < k$ **then**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{S\}$
      update $c$ accordingly
    **else**
      $S_r \leftarrow$ the element $S' \in \mathcal{T}$ that maximises $\sum_{\{S_1,S_2\} \subset (\mathcal{T} \cup \{S\}) \setminus \{S'\}} d(S_1, S_2)$
      **if** $\sum_{\{S_1,S_2\} \subset (\mathcal{T} \cup \{S\}) \setminus \{S_r\}} d(S_1, S_2) \geq c$ **then**
        $\mathcal{T} \leftarrow (\mathcal{T} \cup \{S\}) \setminus \{S_r\}$
        update $c$ accordingly
  **return** $\mathcal{T}$

**Clustering of Space**

Another possible approach is to minimise the average distance between points in $\mathcal{S}$ to its closest point in our sample $\mathcal{T}$, i.e. to find

$$\underset{\mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| \leq k}{\operatorname{argmax}} \quad \frac{1}{n} \sum_{S \in \mathcal{S}} \max_{S' \in \mathcal{T} \cup \{S_0\}} [d(S_0, S) - d(S', S)]$$

where $S_0$ is some arbitrary point in the sentence space. In this section, we will refer to this objective as the CLUSTER objective. This problem is an instance of the submodular maximisation problem. The algorithms to solve such problems will be discussed further in Section 3.

For the purpose of our experiments, we have adapted the P-PASS submodular algorithm presented in [NFTM$^+$18] to solve our CLUSTER objective. This algorithm is a multi-pass streaming-based algorithm which is suited for handling larger amount of data. We present an adaptation of of these ideas in Algorithm 2, where we have fixed some of the tunable parameters and made sure the output solution has size $k$. While

the algorithm can theoretically get stuck in an infinite loop, we have had none of such issues on actual dataset tested. To further save time, we can also pre-compute the distances between each sentences in the dataset, so that we do not have to recompute them again when needed.

**Algorithm 2** Algorithm to diversify for the CLUSTER objective, adapted from the submodular maximisation algorithm presented in [NFTM$^+$18]

**function** DIVERSIFY-CLUSTER($\mathcal{S}, d, k$)
  $\varepsilon \leftarrow 0.1$ ▷ adjustable algorithm parameter
  $T \leftarrow 2/3$ ▷ adjustable algorithm parameter
  define function $f_c : 2^{\mathcal{S}} \to R$ as $f(\mathcal{T}) = \frac{1}{n} \sum_{S \in \mathcal{S}} \max_{S' \in \mathcal{T} \cup \{S_0\}} [d(S_0, S) - d(S', S)]$
  $\Delta^* = \arg\max_{S \in \mathcal{S}} f_c(\{S\})$
  $\mathcal{O} = \{(1+\varepsilon)^i : i \in \mathbb{Z}, \Delta^* \leq (1+\varepsilon)^i \leq k\Delta^*/T\}$
  **for** $v$ in $\mathcal{O}$ **do**
    $\mathcal{T}_v \leftarrow \emptyset$
    $r \leftarrow 1$
    **while** $|\mathcal{T}_v| < k$ **do** ▷ unlike the original algorithm we do not terminate until $\mathcal{T}_v$ has size $k$
      **for** $S$ in $\mathcal{S} \setminus \mathcal{T}_v$ **do**
        **if** $f(\mathcal{T}_v \cup \{S\}) - f(\mathcal{T}_v) \geq T^r \Delta^*/k$ and $|\mathcal{T}_v| < k$ **then**
          $\mathcal{T}_v \leftarrow \mathcal{T}_v \cup \{S\}$
      $r \leftarrow r + 1$
  $v' = \arg\max_{v \in \mathcal{O}} \mathcal{T}_v$
  **return** $\mathcal{T}_{v'}$

### Selecting Dissimilar Points

The final approach considered is to select points which are as dissimilar to each other as possible. Mathematically, this is equivalent to finding

$$\operatorname*{argmax}_{\mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| = k} \quad \min_{S_1, S_2 \in \mathcal{T}; S_1 \neq S_2} \quad d(S_1, S_2).$$

Throughout this work, we will refer to this objective as the DISSIMILAR objective. There are many algorithms of solving this objective. An example of such is the algorithm for the DISC diversity presented in [DP12], which not only tries to find a dissimilar subset, but also makes sure that all of the points in space are within some distance away from at least one of the selected points.

In our tests, however, we will be using a different algorithm, as presented in Algorithm 3. Here, we greedily keep picking new elements into out solution set as long as it is at least some distance away

from the selected sentences. In Line 10, we have to relax the threshold of distance by some amount, which is needed when no other sentences remaining can be added. For our experiments, we have chosen to relax the threshold by an amount proportional to the range of the pairwise distance of the dataset (which can be approximated through sampling of the dataset).

**Algorithm 3** An algorithm to find a subset of points to diversify for the DISSIMILAR objective

```
1:  function DIVERSIFY-DISSIMILAR(S, d, k)
2:      T ← ∅
3:      τ ← an estimate for the maximum distance between two points in S
4:      while |T| < k do
5:          for S in S do
6:              if S ∉ T and d(S, S') ≥ τ for all S' ∈ T then
7:                  T ← T ∪ {S}
8:                  if |T| = k then
9:                      return T
10:         reduce τ by some amount
```

**Handling Large Amounts of Data**

For meaningful training on machine translation model, the number of sentences encountered may be in the order of $10^5$ or above. We therefore have to consider methods to perform diversification on large amounts of data. To account for this, we try to use algorithms which work on a streaming setting. These algorithms are designed such that it looks up the data in a sequential manner, and will not require access to all of the data points all at once.

Another idea that we have to is to perform sampling tasks in batches of the full dataset. We can load a fraction of the full dataset (possibly even load all onto CPU), and diversify the loaded subset of data. Then, we can combine the selected data from each batch to form our final diversified subset. We formalise this idea in Algorithm 4.

## 2.3 Experiments

In this section, we describe the experimental setup for our diversification tests.

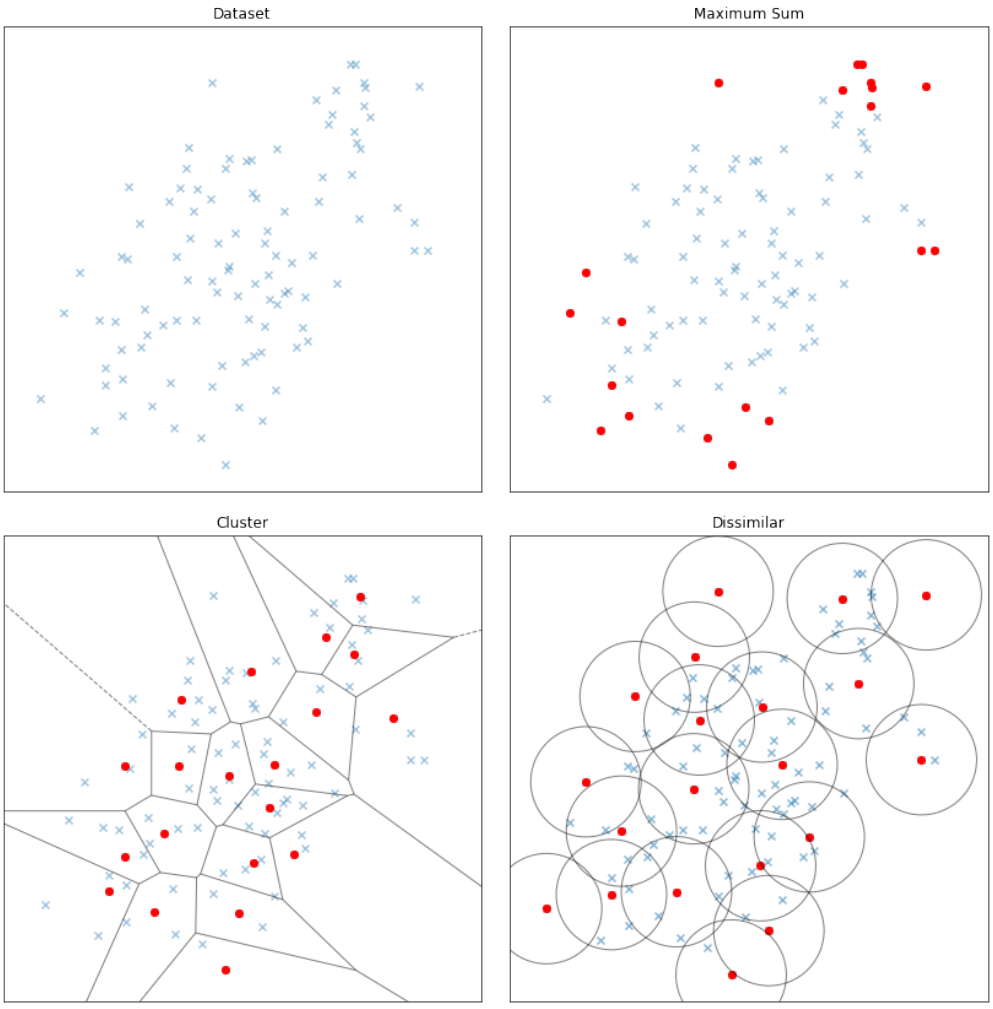


Figure 2.1: Geometric visualisation of the diversification objectives on two-dimensional space. *Top Left:* original dataset. *Top Right:* diversification from the MAXSUM objective. *Bottom Left:* diversification from the CLUSTER objective, where we also partition space using a Voronoi diagram to further demonstrate the representation of selected points. *Bottom Right:* diversification from the DISSIMILAR objective, where a surrounding radius is drawn for each selected point to show that all selected points are at least some fixed distance away from each other.

**Algorithm 4** Diversifying on the dataset in batches

**function** DIVERSIFY-IN-BATCH($\mathcal{S}, d, k$)
  define $b$ to be the number of batches to divide the data into
  $k' \leftarrow k/b$
  $\mathcal{P} \leftarrow$ partition of $\mathcal{S}$ with size $b$ ▷ sets in partition should have roughly equal sizes
  $\mathcal{T} \leftarrow \emptyset$
  **for** $\mathcal{S}'$ in $\mathcal{P}$ **do**
    $\mathcal{T}' \leftarrow$ DIVERSIFY($\mathcal{S}', d, k'$) ▷ diversify using algorithms discussed earlier
    $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$
  **return** $\mathcal{T}$

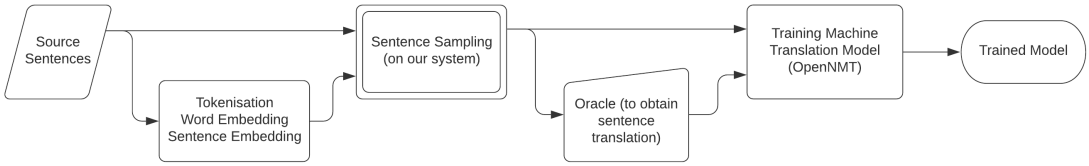


Figure 2.2: The pipeline for the sampling and training process.

### 2.3.1 Pipeline

The flow of the pipeline is presented in Figure 2.2. Given a set of source sentences obtained in some method, they will first be passed into a system to tokenise and/or embed the sentences into data which can be used later on during the sampling stage. For our sampling tasks, we have developed a system for sampling sentences. The system allows for specifying the number of sampled sentences, batch sizes, and for distances and algorithms used for sampling. The entire sampling system has been implemented in Python.

After sampling, we would have to translated the selected sentences. In real life, this could involve human translators who will translate the sentences. In our case however, we will be running experiments on existing parallel corpora, and so the step of "translating" the samples will be replaced by looking up the corresponding translation from the given corpus.

Then, we feed the selected sentences and into a translation model for training. Translation model training is done on the OpenNMT project, an open-source system for neural machine translation [KKD$^+$18]. OpenNMT is a Python-based system, with versions implemented in both Tensorflow and in PyTorch. For our experiment we have used the PyTorch implementation of the system.

### 2.3.2 Diversification Methods Tested

We will be performing sampling tasks using the following distance functions.

- *Jaccard distance on tokens*.
- *Edit distance on tokens*.
- *Normalised edit distance on tokens*.
- *Relaxed Word Mover's distance on token embeddings*. This will be used instead of Word Mover's Distance since it is computationally cheaper. Despite of its relaxation, we found that the distance is still expensive to compute, and some of the tests for this distance has been skipped to save time. The sentences will be embedded using BERT before distance computations.
- *Angular distance on sentence embeddings*. The sentences will be embedded using the Universal Sentence Encoder before distance computations.

We will also be performing the sampling on different diversification objectives that have been mentioned, which includes the following.

- The MAXSUM objective, which is solved using Algorithm 1.
- The CLUSTER objective, which is solved using Algorithm 2.
- The DISSIMILAR objective, which is solved using Algorithm 3.

To work with large number of sentences in the pool of data, we will also be processing the full dataset in batches. As a baseline, we will use batched random sampling to sample the sentences.

### 2.3.3 Corpora

The corpora which are used for our tests were taken from Open Parallel Corpus (OPUS) [Tie12]. We have chosen corpora whose sentence pairs count is in the order of $10^5$. In all of the corpora used, we have

the source sentences be in English, since embedding models are more available in English. Specifically, we used the English - Spanish News corpus, which contained around 250,000 parallel sentences. Using this corpus, we tested all of the available combinations of sampling techniques.

### 2.3.4 Setup For Machine Translation Model

We would like to test how different sampling algorithms and different similarity measures affects the quality of training sets produced. For the corpus tested, we randomly selected 10% of the sentences for evaluating the final model. The remaining 90% of the sentences acts as the pool of sentences we perform our sampling on. During the sampling phase, we first randomly select around 10,000 sentences to be the validation set, then amongst the remaining sentences, we sample out either around 10%, 25% or 50% of the sentences in the pool. These sentences are sampled out using different algorithms and similarity measures, which are described previously. The selected sentences then act as the training set for our model. The sentences then undergoes the process of Byte Pair Encoding (BPE) which reduces required vocabulary size of the corpus by breaking down larger, infrequent words into multiple tokens which are more frequent in the corpus [SHB15]. For our experiment, we aim for the vocabulary size to be at around 30,000.

The machine translation tasks are done on models with a transformer architecture. The parameters for the model are adapted from the default values from OpenNMT and from some of the models tested in [VSP+17]. We have listed these parameters in Table 2.1. All of the training is performed on a single GPU.

Since some training sessions are done on small number of sentences, methods to reduce over-fitting are needed. We have done so by slightly increasing dropout rate (compared to the papers we referenced), and by allowing for early termination if the validation accuracy or validation perplexity does not improve after some number of validation steps. After allowing for early termination, we found that all of the training done terminates before the maximum number of steps are reached, and most of the models only took a few hours to train until this point.

During our experiments, we also found that OpenNMT will often run into out-of-memory errors if our validation batch size is too high. We therefore were only able to set the validation batch size to a small

| Parameter | Model Value |
|---|---|
| **Model Parameters** | |
| Number of Layers | 2 |
| Word Vector Size | 512 |
| Sinusoidal Position Encoding | True |
| Feed-Forward Size | 2048 |
| Heads | 8 |
| | |
| **Training Parameters** | |
| Batch Size | 4096 |
| Gradient Accumulation | 2 |
| Maximum Steps | 100000 |
| Validation Frequency | Every 1000 Steps |
| Early Termination | 10 Validation Steps |
| Dropout Rate | 0.2 |
| | |
| **Optimiser Parameters** | |
| Type | Adam |
| Warm-up Steps | 8000 |
| Initial Learning Rate | 2 |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.998 |

Table 2.1: Parameters for Machine Translation Model

number (in our case, we have set it to 4). A small validation batch size had negligible effect to the overall time required for training, and no effect on the quality of output translation.

After training is completed, we translate our tests and de-tokenise them accordingly. The quality of translation is evaluated using the BLEU score, which is based on whether the translated sentence has the $n$-grams that appears in the reference sentence (the actual translation) or not. The use of $n$-grams makes sure that a good translation not only has the correct words, but that those words also appear in a grammatically correct order, or at least compared to the reference [PRWZ02].

**Analysis of Sampled Sentences**

In addition to testing the effects of sampling techniques on quality of the trained model, we would also like to compare the similarity between the sampled sentences. We will also be comparing the overlap between selected subsets due to different techniques, and also the lengths of the selected sentences.

Furthermore, we wanted to analyse the change in the distribution of pairwise sentence distances. To do so, we created a mock dataset only consisting of 10,000 sentences, and performed sampling to obtain 1,000 sentences. Using the sampled 1,000 sentences, we found the distance between every sentence pair and recorded the data. We did so for every sampling algorithm and every distance metric we have.

## 2.4 Results and Analysis

Below we present and analyse the data collected from the experiments.

### 2.4.1 Performance of Sampled Set in Training Translation Models

The training results are summarised in Figure 2.3. Disappointingly, we found that in most of the combinations of objective and distance metrics we have tested results in sampled data which is detrimental to the training process.

Sampling under the MAXSUM objective consistently results in training sets which creates bad translation models, in cases creating models which lowers the BLEU index by as much as 4. Intuitively, this may be due to the fact that it tends to select samples which are as far from each other as possible, and as a result tends to select the samples at the "edge" of the corpus. On the other hand, sampling under the CLUSTER objective or the DISSIMILAR objective may result in good samples, depending on the distance metric used.

Edit distance tends to always produce good training results regardless of the diversification objective. However, we predict that this could be due to the fact that when diversifying using the edit distance the longer sentences are being favoured (we will discuss this point in the following subsection). We believe that a longer sentence results in larger number of tokens for the model to train with, which is similar to having more training samples to work with.

Among the other two tokens-based distances the results are somewhat promising. In our tests, we find that normalised edit distance consistently manages to outperform our random benchmark when sampled under the DISSIMILAR objective by increasing the testing BLEU score by around 2 points, while

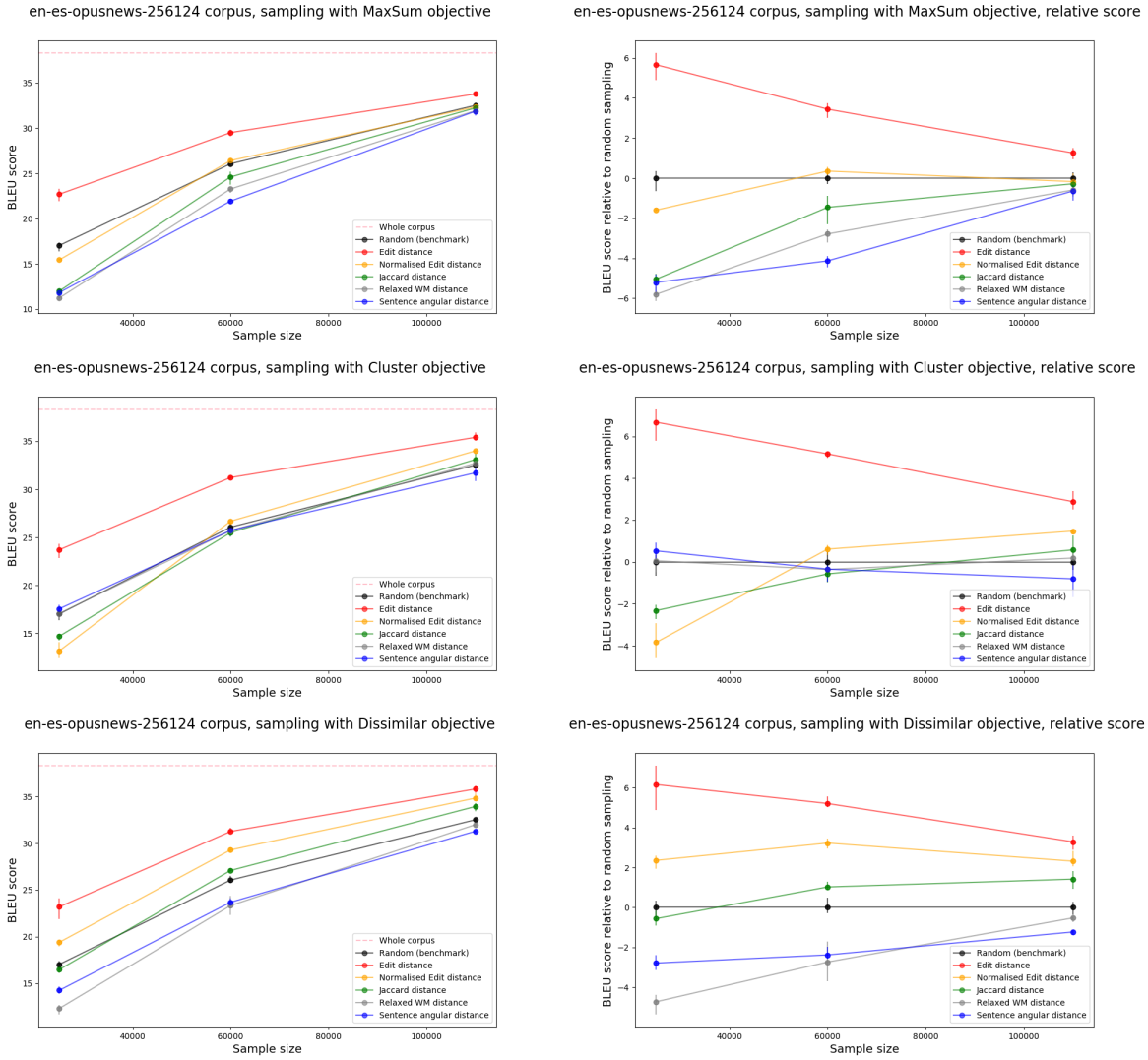


Figure 2.3: BLEU scores from training on different diversified training set. For each distance function, we sampled the English-Spanish corpus to get diversified training set of different sizes. The graphs on the left column shows the BLEU score from testing, and the graphs on the right column are the same scores relative to the random benchmark done. The random data in all graphs are from the same training instance. All of the error bars represent the minimum and maximum of performed tests. *Top:* sampling done using the MAXSUM objective. *Middle:* sampling done using the CLUSTER objective. *Bottom:* sampling done using the DISSIMILAR objective.

only slightly improving from random benchmark when sampled under the CLUSTER objective. For Jaccard distance, we see that it only slightly improves the testing BLEU scores by less than 1 in the DISSIMILAR objective, which is too low of an improvement to draw any conclusions.

We also found that the two distance metrics based on different embeddings perform poorly in our tests. Both relaxed Word Mover's distance an cosine distance on sentence embeddings result lowers training scores compared to our random benchmark by up to 2-4 BLEU points in some instances. The reasons for this may be due to our decision to not fine-tune the embedding models we have used, or the fact that only considering sentences at token level is inherently better than performing sampling on context-based sentence representations.

### 2.4.2 Analysis of the Sampled Subsets

#### Similarities of Sampled Subsets

In addition to performing training on the sampled subsets, we also wanted to perform further analysis on our sampling techniques. We first wanted to investigate the similarities between different sampling techniques by seeing how similar the selected subsets are compared to each other. To do this, we compared each sampling techniques against each other and found what proportions of sentences were selected in common between these sampling techniques. We present these results in Figures 2.4, 2.5 and 2.6.

From the similarity plots, we find that sampling using the MAXSUM objective and DISSIMILAR objective tend to produce more similar subsets. For sampling instances when the same distance metrics are used, sampling using the two objectives can give up to 10% more overlap compared to the random sampling case. This is in contrast to the sentences sampled using the CLUSTERING objective which will give less overlap when compared to the other objectives.

We can also see the overlap of selected sentences when we sample using normalised edit distance, Jaccard distance and relaxed Word Mover's distance. We can see that when sampling using DISSIMILAR or MAXSUM objectives, we see a significant increase in sample overlaps when we use normalised edit distance, Jaccard distance and relaxed Word Mover's distance as our similarity metric.

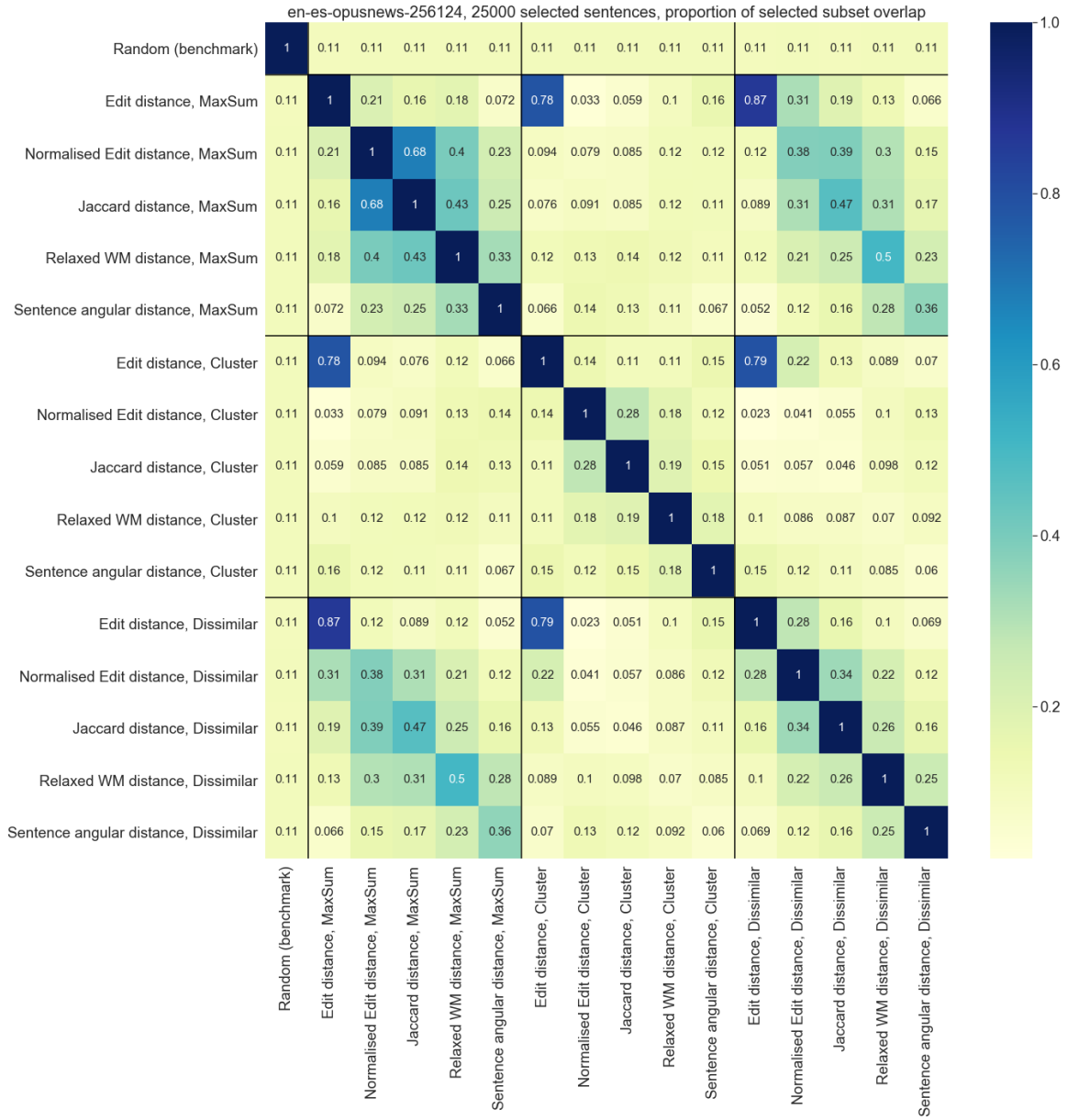


Figure 2.4: Heat-map to demonstrate the similarity between the selected subsets from different methods. The number represents the proportion of selected samples which are shared between the particular sampling instances listed by row and column. All sampling instances here selected 25,000 sentences from the whole corpus.

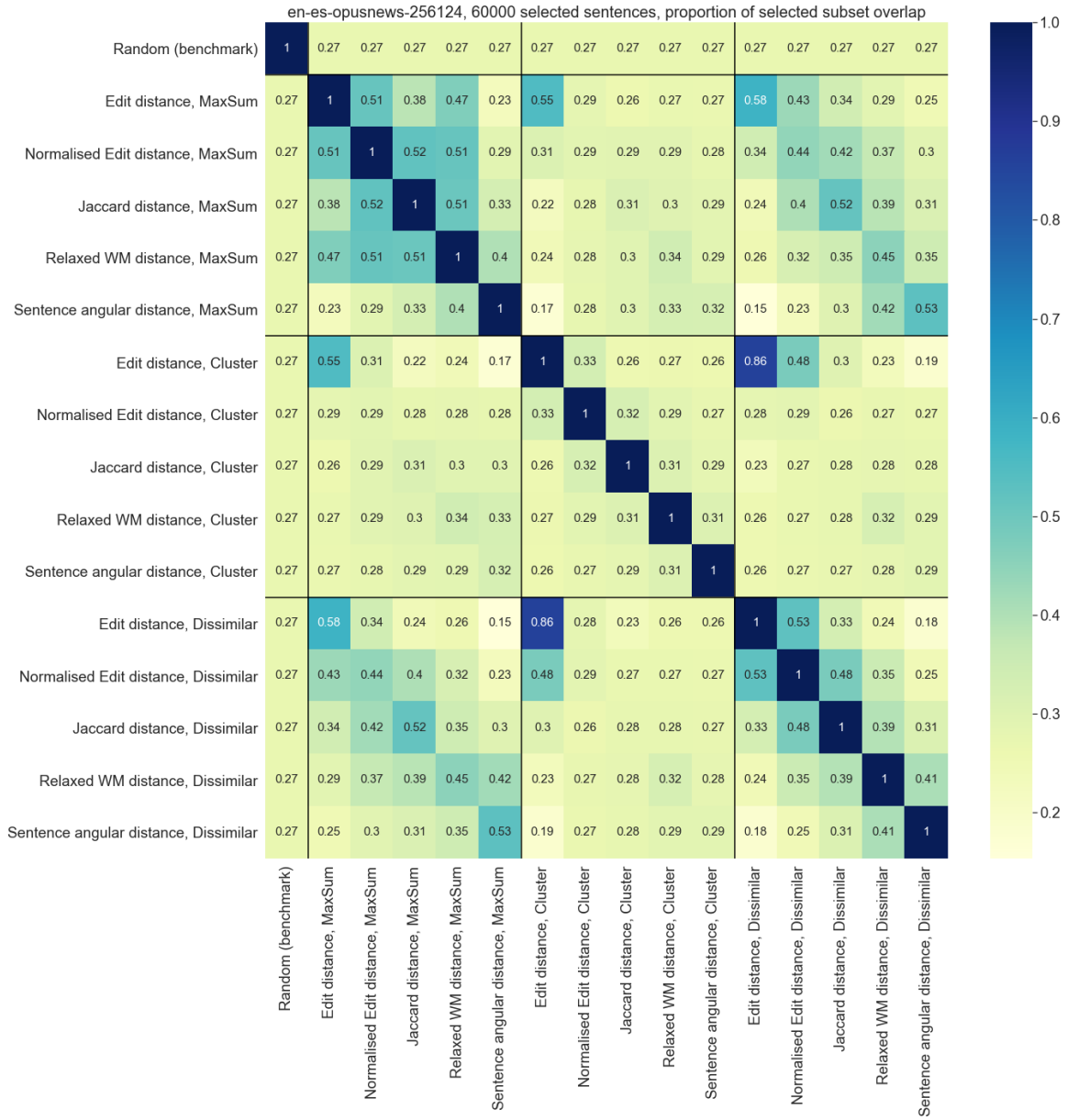


Figure 2.5: Heat-map to demonstrate the similarity between the selected subsets from different methods. The number represents the proportion of selected samples which are shared between the particular sampling instances listed by row and column. All sampling instances here selected 60,000 sentences from the whole corpus.

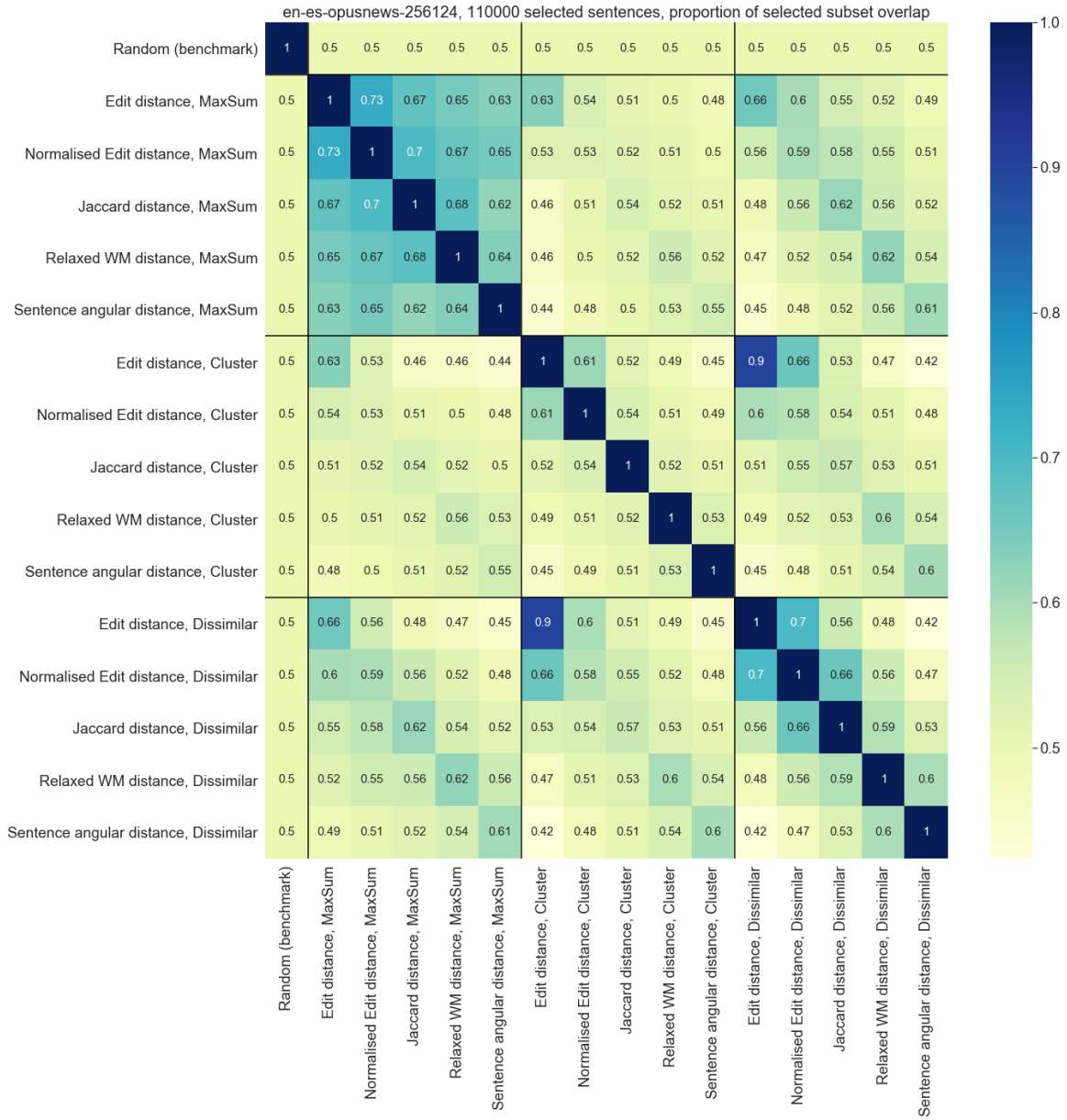


Figure 2.6: Heat-map to demonstrate the similarity between the selected subsets from different methods. The number represents the proportion of selected samples which are shared between the particular sampling instances listed by row and column. In each plot, we give the mean, standard deviation and skew of the distance distribution. All sampling instances here selected 110,000 sentences from the whole corpus.

**Lengths of Sampled Subset**

In Figure 2.7, we plotted the distribution of the token lengths of the selected subsets from the different sampling techniques. Here, we found that the sampling techniques that gave good results tend to give subsets which had longer sentences. We can see that the sampling techniques which gave significant increase in average sentence lengths when compared to random sampling were samplings done on the edit distance metrics and for sampling using the DISSIMILAR metric on normalised edit distance. These sampling instances also corresponded to the samples which gave better training scores.

**Distribution of Pairwise Distances**

In Figures 2.8, 2.9 and 2.10, we have plotted the distribution of distance between sentences in the sampled subset from our smaller mock dataset. Trends similar to that presented in the subset overlap data can be seen here.

For both the MAXSUM and the DISSIMILAR objective, we find that normalised edit distance, Jaccard distance and relaxed Word Mover's distance tend to have similar effects on the selected subset. From the graphs, we can see that sampling using either of the three metrics will result in an increase in the mean value of the distance distributions which are based on normalised edit distance, Jaccard distance and relaxed Word Mover's distance. This provides further evidence on the correlation amongst these three distance metrics.

## 2.5 Conclusion

In this section, we have explored different methods to perform sentence diversification for training set reduction for machine translation tasks. We have developed a pipeline for sampling sentences from a large corpus using various metrics and algorithms, and compared the quality of the sampled subset by testing their performance towards training a machine translation model.

From the results obtained from the pipeline, we can see that some similarity measures and some diversification techniques were able to give improvements in the training data, while some can be detrimental

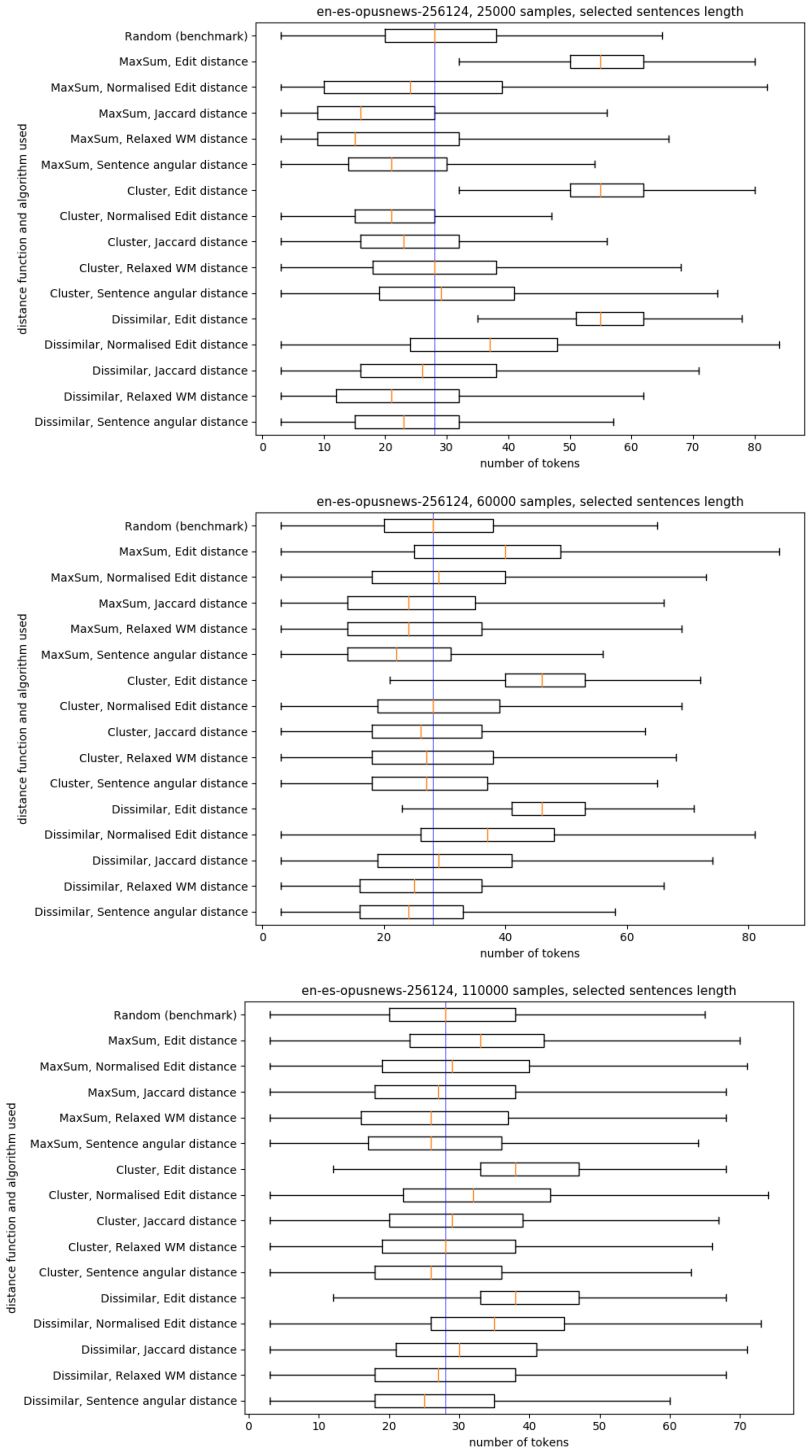


Figure 2.7: Box-plots to present the (token) length of each sampled subset. The blue line in each graph represents the median from random sampling. From top to bottom, the data is from sampling subset of sizes 25,000, 60,000, and 110,000.

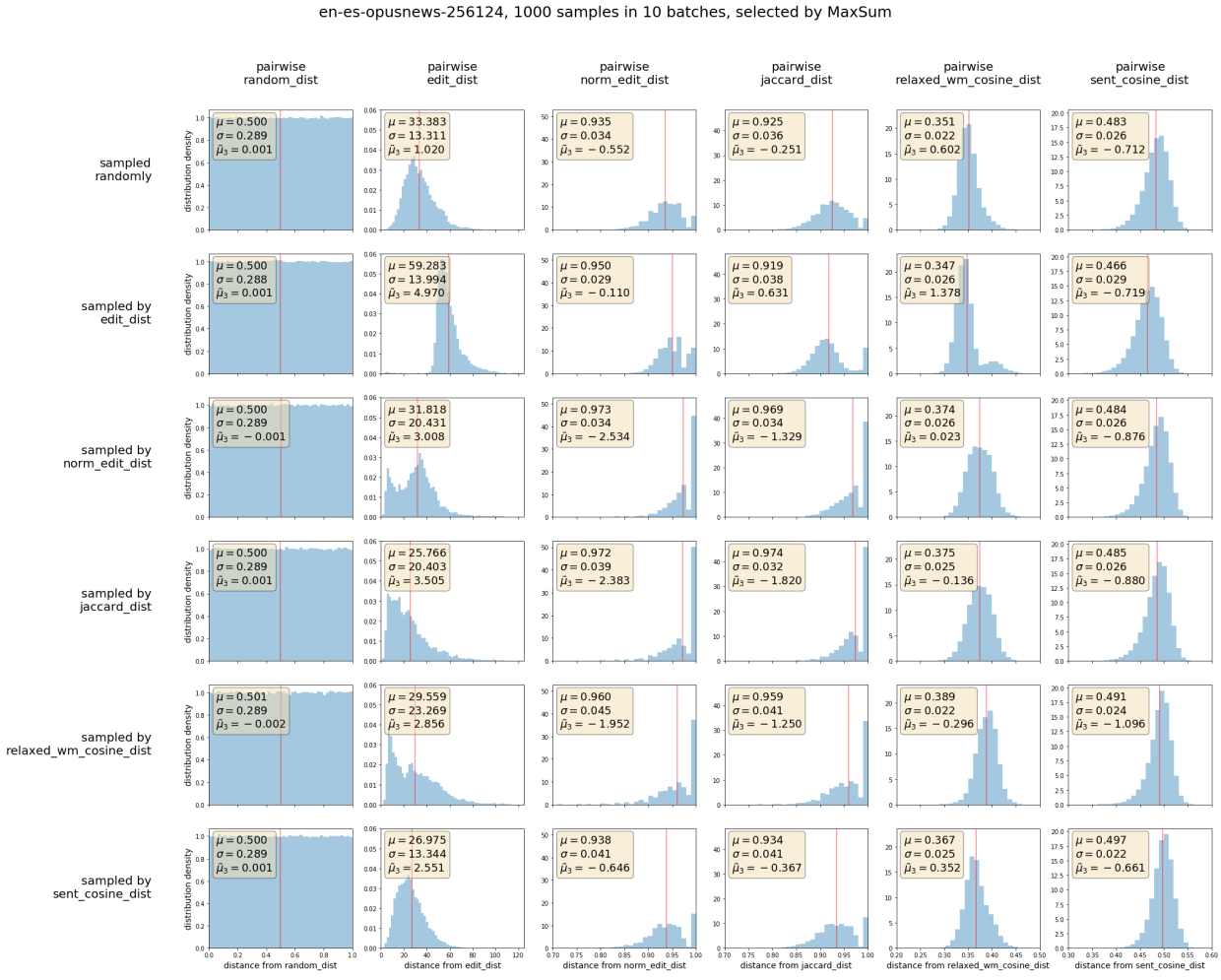


Figure 2.8: Distribution of pairwise distances on a subset of 1,000 sentences sampled from a mock dataset of size 10,000. Plots in the same row are the data from the sentences sampled using the same distance metric, and plots in the same column are pairwise distances measured on the same metric. In each plot, we give the mean, standard deviation and skew of the distance distribution. All samples are done on the MaxSum objective.

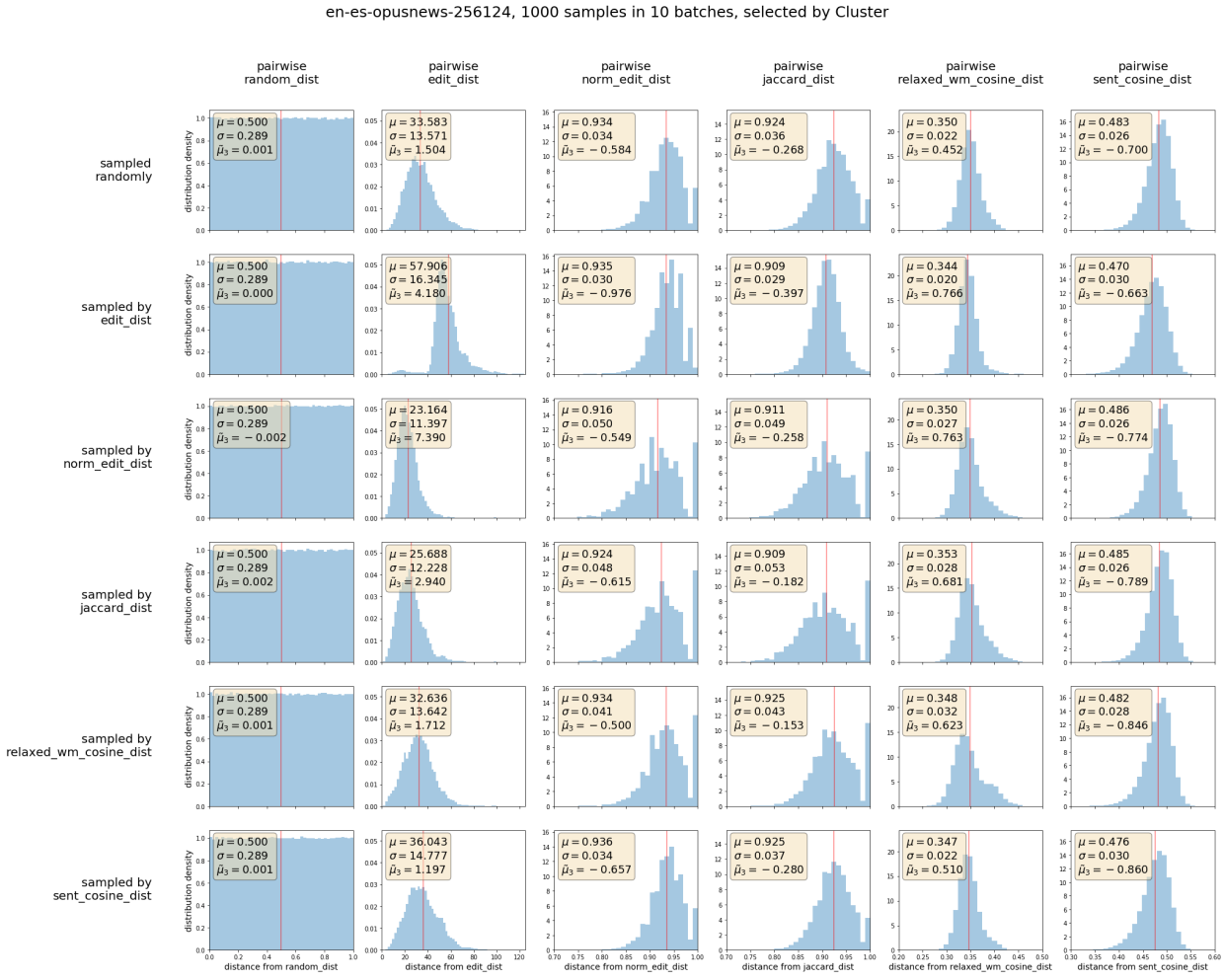


Figure 2.9: Distribution of pairwise distances on a subset of 1,000 sentences sampled from a mock dataset of size 10,000. Plots in the same row are the data from the sentences sampled using the same distance metric, and plots in the same column are pairwise distances measured on the same metric. In each plot, we give the mean, standard deviation and skew of the distance distribution. All samples are done on the CLUSTER objective

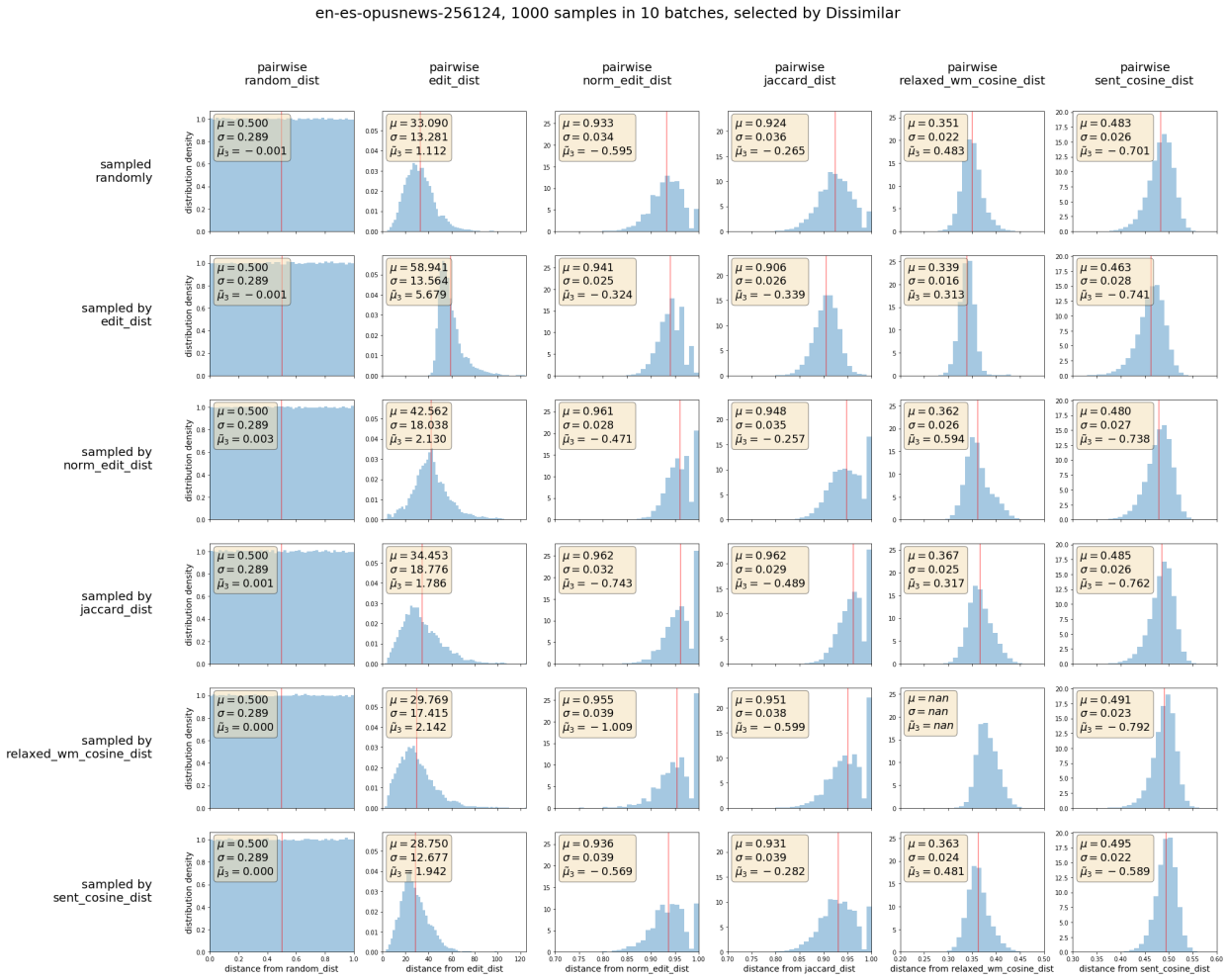


Figure 2.10: Distribution of pairwise distances on a subset of 1,000 sentences sampled from a mock dataset of size 10,000. Plots in the same row are the data from the sentences sampled using the same distance metric, and plots in the same column are pairwise distances measured on the same metric. All samples are done on the DISSIMILAR objective

to the results.

There still remains a large number of remaining work that can be done, including the following

- We can still attempt to perform similar tests on other corpuses, and to see how the results will vary. Also, we can run experiments on larger corpuses to see how well our framework scales.
- We can attempt to understand the results we have obtained further. We are still uncertain about why edit distance always performs better than the other metrics, or why embedding-based distances performs poorly. While we have hypotheses about these, more analysis would be needed to arrive at a conclusion.
- We can find other methods to perform similarity measures. For example, we can look at performing diversification using similar distance metrics but without considering words which are common in the corpus (e.g. in English, these may include the articles or common pronouns).

# CHAPTER 3

# AN EFFICIENT PARALLEL SUBMODULAR MAXIMISATION ALGORITHM

## 3.1 Introduction

In this chapter, we will be building up towards an efficient submodular maximisation algorithm. We will start by introducing the idea of submodular functions in Section 3.2, where we will talk about its definition, its application to other problems in machine learning, and methods of solving them in different settings. Then, in Section 3.3, we will further discuss some of the useful subroutines that have arisen from previous works on submodular maximisation algorithms. In Section 3.4, we will present some problems of existing submodular maximisation algorithms, and propose our own submodular maximisation algorithm which will address some of these concerns. Finally, we will test our proposed algorithm against different benchmarks using methods as noted in Section 3.5, and present the results in Section 3.6.

## 3.2 Background Information

In this section, we will describe the problem of submodular maximisation, and examine existing methods that have been proposed for different settings.

### 3.2.1 Submodular Maximisation Problem

Submodular functions are an important class of functions which can be used to model many interesting optimisation problem. First, we present its definition below.

**Definition 3.2.1** (Submodular Functions)**.** *For any set $V$, a function $f : 2^V \to \mathbb{R}$ is submodular if, for any subsets $S \subseteq T \subseteq V$, it follows that*

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T).$$

It is useful to contrast this with *modular* functions, which are functions where $f(S) + f(T) = f(S \cup T) + f(S \cap T)$ for any subset $S \subseteq T \subseteq V$. Another useful definition is the *marginal gain* of a function.

**Definition 3.2.2** (Marginal Gains of Functions)**.** *Let $V$ be any set, and define $f : 2^V \to \mathbb{R}$ be a function. For any element $e \in V$ and any subset $S \subseteq V$, we define the marginal gain of $e$ to $S$ to be $f(e|S) := f(S \cup \{e\}) - f(S)$. Similarly, for any subsets $S, T \subseteq V$, we define the marginal gain of $T$ to $S$ to be $f(T|S) := f(S \cup T) - f(S)$.*

Marginal gain reflects the increase in the function's value when an element or elements are added into the set. For *monotone* submodular functions, marginal gains will always be non-negative.

From the definition of submodular functions, we arrive at an important property of submodular functions.

**Theorem 3.2.1.** *Let $V$ be any set, and define a submodular function $f : 2^V \to \mathbb{R}$. For any element $e \in V$ and for any subsets $S \subseteq T \subseteq V$, it follows that $f(e|S) \geq f(e|T)$.*

Theorem 3.2.1 is perhaps a much more intuitive definition for submodular functions. In simple terms, a submodular function captures the idea of diminishing returns. Adding an element into a large set will give a smaller increase in the function value than if we were to add the same element into a smaller set, since adding a new element into a larger set will present the set with less new "information".

A particular interesting type of problem is where we try to maximise a particular submodular function given some constraints. In the literature, submodular maximisation under Knapsack constraints or matroid constraints have been studied. However, in this work, we will be mostly concerned with maximising submodular functions under cardinality constraint, which formally, is an optimisation problem where we want to

$$\text{maximise } f(S) \text{ for some } S \subseteq V \quad \text{subject to} \quad |S| \leq k$$

for some submodular function $f$ and some natural number $k$. Here, $k$ denotes the maximum cardinality of the optimal set.

### 3.2.2 Applications of Submodular Functions

Submodular functions are interesting because they are a good generalisation for many problems we find across machine learning. Below, we discuss some of these problems.

#### Exemplar-Based Clustering

In the clustering problem we want to pick a set of points such that it is the most representative of the other points in the pool. To do so, we can select points such that it maximises the closeness to the other points in the pool. Mathematically, we can maximise a submodular function $f : 2^V \to \mathbb{R}$ where

$$f(S) = \frac{1}{|V|} \sum_{e \in V} \max_{a \in S \cup \{e_0\}} [d(e_0, e) - d(a, e)],$$

where $d(x, y)$ is a distance function and $e_0$ is an arbitrary point in space. Intuitively, the function quantifies the average distance between any point in set $V$ to its closest point from the selected set $S$.

#### Active Set Selection

Given some elements $a, b \in V$, we can quantify the "similarity" between the objects as $e^{-d(a,b)}$ where $d(a, b)$ is the distance between the two elements. Using this, we can define a matrix $M$ such that $M_{i,j} = e^{-d(v_i,v_j)}$ for some $v_i, v_j \in V$.

Given a set of elements $S \subseteq V$, we can define a function that quantifies the diversity of the set $S$. Specifically, we can let

$$f(S) = \log \det(I_{|S|} + \alpha M_S)$$

where $M_S$ is the principal sub-matrix of $M$ indexed by $S$ (i.e. matrix $M$ but keeping the rows and columns according to elements in $S$), $I_{|S|}$ is the identity matrix of dimension $|S|$, and $\alpha > 0$ is a regularisation parameter. It can be shown that this function is submodular [LSH02].

#### Recommendation Systems

In recommendation systems, the goal is to recommend a user to a number of diverse objects which they might be interested in. This can be broken down into two sub-objectives of recommending diverse

objects, and recommending objects the user will like.

An example of this is with movie recommendations [NFTM$^+$18]. Suppose $M = \{m_1, m_2, \ldots, m_n\}$ is the sets of movies, and let $u$ be a user we are interested in. Suppose we are able to obtain vector representations for the movies, $v_{m_1}, v_{m_2}, \ldots, v_{m_n} \in \mathbb{R}^p$, and a vector representation for the user $w_u \in \mathbb{R}^p$ such that $\langle v_{m_i}, w_u \rangle$ equals to the rating user $u$ would give for movie $m_i$, and the term $\langle v_{m_i}, v_{m_j} \rangle$ correlates to the degree of similarity between the movies. Then, we can define a submodular function $f : 2^M \to \mathbb{R}$ where

$$f(S) = \alpha \sum_{m' \in M} \max_{m \in S} \langle v_m, v_{m'} \rangle + (1 - \alpha) \sum_{m \in S} \langle w_u, v_m \rangle,$$

where $\alpha$ is a trade-off parameter (for setting how much each objective should be weighed). The first term measures how well the selected subset $S$ covers the space of the object, while the second term measures how much the user will like the objects from $S$.

**Maximum Coverage Problem**

Suppose we have a set $A$. Also suppose $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ is a set of subsets of $A$, i.e. $T_i \subseteq A$ for all $i$. For a set $\mathcal{S} \subseteq \mathcal{T}$, we can define a submodular function $f : 2^{\mathcal{T}} \to \mathbb{R}$ where

$$f(\mathcal{S}) = \left| \bigcup_{T \in \mathcal{S}} T \right|.$$

In other words, $f(\mathcal{S})$ measures how many elements from $A$ is contained in $\mathcal{S}$.

### 3.2.3 Existing Algorithms

In this section, we will explore some algorithms proposed to solve the submodular maximisation problem under cardinality constraints. We will reference prior works, including algorithms which have tackled the submodular maximisation problem under the parallel setting or under the streaming setting.

**The Greedy Algorithm and Its Variance**

The problem of submodular maximisation, unfortunately, is NP-Complete, meaning that no efficient polynomial-time algorithm exists for the exact optimal solution unless P = NP. A simple solution,

however, is to use a greedy algorithm, where we find the element from the ground set with the highest marginal gain to our solution set, and add this new element to our solution set until the maximum cardinality is reached. This idea is presented in Algorithm 5. Interestingly, this algorithm gives an approximate solution with the best possible approximation bound for any polynomial-time algorithm [NWF78]. This result is summarised in Theorem 3.2.2.

**Algorithm 5** Greedy Submodular Maximisation Algorithm

```
function GREEDY-SUBMOD(V, f, k)
    S ← ∅
    V' ← V
    while |S| < k do                ▷ one while loop is said to be one round of the greedy algorithm
        v_m ← arg max_{v∈V'} f(v|S)
        S ← S ∪ {v_m}
        V' ← V' \ {v_m}
    return S
```

**Theorem 3.2.2.** *Let $f : 2^V \to \mathbb{R}$ be a monotone submodular function, and $k$ be the maximum cardinality for the optimal set. Let* OPT *be the optimal value for $f$ under the cardinality constraint, i.e.* $\texttt{OPT} = \max_{S \subseteq V, |S| \leq k} f(S)$. *For a submodular maximisation problem, the greedy algorithm will yield a solution $S'$ such that*

$$f(S') \geq \left(1 - \frac{1}{e}\right) \texttt{OPT}.$$

*Furthermore, no polynomial-time algorithm can provide a better approximation guarantee beyond $1 - 1/e$ unless* P = NP.

*Proof.* We will only prove the approximation bound of the greedy algorithm here, and refer the reader to the original paper by Nemhauser, Wolsey and Fisher [NWF78] for the proof to the claim that no polynomial-time algorithm can perform better than the greedy algorithm. The idea of the approximation bound proof presented here is the same as in the survey paper from Krause and Golovin [KG14].

Let $f : 2^V \to \mathbb{R}$ be a submodular function. Let $S^* = \underset{S \subseteq V, |S| \leq k}{\arg\max} f(S) = \{v_1^*, v_2^*, \ldots, v_k^*\}$, which is the optimal set, and let $\texttt{OPT} = f(S^*)$ which is the optimal value of the function under the cardinality constraint.

Suppose the greedy algorithm returns a final solution $S' = \{v_1, v_2, \ldots, v_k\}$. By construction of our greedy algorithm, it is guaranteed that $|S'| = k$. Let $S'_i = \{v_1, v_2, \ldots, v_i\}$ for some $1 \leq i \leq k$. Similarly, let $S^*_i = \{v^*_1, v^*_2, \ldots, v^*_i\}$ for some $1 \leq i \leq k$. We can see that

$$\begin{aligned}
f(S^*) &\leq f(S^* \cup S'_i) && \text{by submodularity,} \\
&= f(S'_i) + \sum_{j=1}^{k} f(v^*_j | S'_i \cup S^*_{j-1}) \\
&\leq f(S'_i) + \sum_{j=1}^{k} f(v^*_j | S'_i) && \text{by submodularity,} \\
&\leq f(S'_i) + \sum_{j=1}^{k} f(v_{i+1} | S'_i) && \text{since } v_{i+1} = \arg\max_{v \in N \setminus S'_i} f(v|S'_i) \text{ by the algorithm,} \\
&= f(S'_i) + k f(v_{i+1} | S'_i)
\end{aligned}$$

which means that $f(v_{i+1}|S'_i) \geq \frac{1}{k}(\texttt{OPT} - f(S'_i))$.

Let $\delta_i = \texttt{OPT} - f(S'_i)$. We can see that $\delta_i - \delta_{i+1} = f(S'_{i+1}) - f(S'_i) = f(v_{i+1}|S'_i) \leq \delta_i/k$, or that $\delta_{i+1} \geq \left(1 - \frac{1}{k}\right)\delta_i$. This means that $\texttt{OPT} - f(S') = \delta_k \leq \left(1 - \frac{1}{k}\right)^k \delta_0 = \left(1 - \frac{1}{k}\right)^k \texttt{OPT} \leq \texttt{OPT}/e$ since $(1-x)^x \leq 1/e$ for any $x \geq 0$, hence we can conclude that $f(S') \geq \left(1 - \frac{1}{e}\right)\texttt{OPT}$. □

While the greedy algorithm gives very good approximations, it can be extremely slow in practice, especially when there is a lot of data. More efficient algorithms have therefore been proposed, which aim to reduce the amount of time to find a solution, while sacrificing the least amount of accuracy as possible. For example, a simple adjustment which can be made to the greedy algorithm is to use submodularity of our objective function to determine which elements in the ground set definitely cannot be the element with the highest marginal gain without having to make a query call to the oracle. This version of the greedy algorithm is referred to as the LAZY-GREEDY algorithm, which can give large performance boosts compared to the plain greedy algorithm [Min78].

The problem of submodular maximisation on large ground sets have also been studied, which incorporates techniques of parallelism or data streaming into the algorithm as well. We present some of these algorithms below.

**Algorithms in the Parallel Setting**

In the parallel setting, we assume multiple instructions can be executed simultaneously. The algorithms in this settings are evaluated based on their approximation guarantees and their adaptivity, which represents how many rounds of parallel function calls are required for the algorithm. More recent works have produced parallel algorithms with an approximation factor of $1 - 1/e - \varepsilon$, while having an adaptivity of $O(\log n/\varepsilon^2)$, where $\varepsilon$ is the approximation factor which is determined by the user.

In algorithms presented by Balkanski, Rubinstein and Singer [BRS18] and by Ene and Nguyen [EN18], a similar strategy is used. Both algorithms consist of a filtering process which aims to remove elements from the ground set which has low expected contribution to the sample set, and a stage to add a set of random elements into the sample set. In these algorithms, they are able to guarantee that the filtering process will be able to discard a constant fraction of the elements from the ground set, that the randomly selected set is expected to have some amount of marginal gain to the original set, and that the whole process is able to run in some fixed number of rounds. In addition to these techniques, since the algorithms tend to require us to know the value of `OPT`, multiple copies of the algorithm in parallel are run, where each copy has its own guess of `OPT`.

In addition to the parallel algorithms from above, the algorithm presented by Fahrbach, Mirrokni and Zadimoghaddam [FMZ18b] also guarantees (in expectation) the same approximation and adaptivity, however only requiring $O(n)$ function calls. In addition to the filtering process and repeatedly adding new random subsets into the sampled set, the algorithm also deploys methods to reduce the approximation bounds for `OPT`, so that not as many parallel copies of the submodular maximisation have to be run for as many approximations of `OPT`.

While the parallel algorithms presented in the literature works well in theory, there has been no evidence (to our knowledge) of the algorithms being used on real sample sets. It is also important to point out that the algorithm may not scale well despite of low adaptivity, as a large number of functions may not be able to be executed simultaneously in real life since it is not possible to guarantee that an infinite number

of tasks can be executed concurrently in real life.

There have also been studies for submodular maximisation algorithms under a distributed setting. The algorithm here are also able to execute instructions in parallel, however it will also try to reduce the amount of information that each worker shares with each other since each workers may be in different machines, and communication between them will consume a larger amount of time. Algorithms under this setting has been developed, many using the MAPREDUCE regime [KMVV15].

**Algorithms in the Streaming Setting**

In the streaming setting, we assume that the data $\{e_1, e_2, \ldots, e_n\}$ cannot all be accessed at the same time, but rather the elements arrive to the algorithm one-by-one, and only $o(n)$ of those elements can be stored by the algorithm. In the one-pass setting, any data $e_i$ which has been discarded by the algorithm will be lost forever and cannot be returned, so the moment the algorithm sees the element $e_i$, it would have to decide on whether to keep it in memory or discard it. Multiple one-pass streaming algorithms have been proposed which all achieve approximation guarantees arbitrarily close to $1/2$ [BMKK14, NFTM$^+$18, KMZ$^+$19]. Additionally, better approximation guarantees are possible for streaming algorithms which allow more than one passes over the entire data stream [NFTM$^+$18].

**Algorithms For Other Constraints**

There have also been algorithms proposed for solving monotone submodular maximisation problems with other constraints as well, for example, submodular maximisation under matroid constraints. Parallel algorithms with adaptivity of $O(\log^2 n/\varepsilon^3)$ have been proposed to solve such algorithms [CQ18]. Additionally, there have also been parallel algorithms proposed for *non*-monotone submodular maximisation as well [FMZ18a].

## 3.3 Subroutines From Previous Works Used

In the previous section, we have provided an overview of some of the existing algorithms in the parallel setting. We will now examine some of these algorithms more closely.

### 3.3.1 The THRESHOLD-SAMPLING Algorithm

THRESHOLD-SAMPLING is one of the subroutines presented by Kazemi et al [KMZ+19]. The algorithm aims to build a set by selecting a number of elements whose average marginal gain is above a certain threshold $\tau$. However, this constraint may be relaxed, and so an error term $\varepsilon$ is introduced.

In a round of THRESHOLD-SAMPLING, we start by filtering out the elements whose marginal gains are lower than the required threshold using the FILTER subroutine. Then, using the remaining elements, we sample out some of them and add them to our solution if their marginal gain is high enough. We do so until we hit the maximum allowed cardinality or we run out of elements to randomise from. Using this, we are able to guarantee that the elements we have added into our set will contribute some amount of marginal gain, and that the elements discarded definitely have marginal gains which are too low for our solution. The pseudocode for these algorithms are presented in Algorithms 6 and 7, both which are adaptations from the paper by Kazemi et al., and summarise their performance in Theorem 3.3.1.

**Theorem 3.3.1.** *The algorithm* THRESHOLD-SAMPLING *returns a set $S \subseteq V$ such that*

1. *The expected number of queries to the submodular function is $O(|V|/\varepsilon)$,*
2. *The average marginal gain for each element $f(S)/|S|$ is greater than $(1-2\varepsilon)\tau$, and*
3. *If $|S| < k$, then $f(x|S) < \tau$ for all $x \in V \setminus S$.*

To prove this, we will make use of the following result from statistics.

**Theorem 3.3.2** (Hoeffding's Inequality [Hoe63])**.** *Let $X = \sum_{i=1}^{n} X_i$ where each $X_i$'s are random variables which are all independently distributed in $[0,1]$. Let $\mu = \mathbb{E}[X]$. Then, for all $t > 0$, $\Pr[X > \mu + t]$ and $\Pr[X < \mu - t]$ is at most $\exp(-2t^2/n)$.*

Using this inequality, we will now prove Theorem 3.3.1.

*Proof.* We will prove the three claims in the theorem separately. Note that the first point is a consequence of results shown in the original paper, whereas the second and third points are results directly from the paper (hence their proof has already been stated in the paper but will also be mentioned here for completeness).

**Algorithm 6** THRESHOLD-SAMPLING subroutine

```
1: function THRESHOLD-SAMPLING(V, f, k, τ, ε)
2:     S ← ∅
3:     V' ← V
4:     while |S| < k and |V'| > 0 do
5:         V' ← FILTER(V', f, S, τ)
6:         for ⌈1/ε⌉ loops do
7:             x ← a random element from V'
8:             if f(x|S) ≤ (1 − ε)τ then
9:                 break and go to Line 2
10:            else
11:                S ← S ∪ {x}
12:                V' ← V' \ {x}
13:            if |S| = k or |V'| = 0 then return S
14:        for i = ⌊log_{1+ε}(1/ε)⌋ to ⌈log_{1+ε} k⌉ do
15:            t ← min{(1 + ε)^{i+1} − (1 + ε)^i, |V'|, k − |S|}
16:            T ← a random subset of size t from V'
17:            if f(T|S) ≥ (1 − ε)τ · |T| and |S ∪ T| < k then
18:                S ← S ∪ T
19:                V' ← V' \ T
20:            else
21:                S ← S ∪ T
22:                V' ← V' \ T
23:                break
24:    return S
```

**Algorithm 7** Filtering Algorithm

```
1: function FILTER(V, f, S, τ)
2:     for v in V do
3:         if f(v|S) ≤ τ then
4:             V ← V \ {v}                ▷ Remove v from set
5:     return V
```

To prove the first claim, we will first show that the size of set $V'$ is expected to decrease with a constant fraction in each iteration. Let $V'_0$ be the set $V'$ initially, and $V'_r$ be the set $V'$ after $r$ iterations of the while loop. We claim that $|V'_{r+1}| \leq (1-\varepsilon/2)|V'_r|$ with a constant probability. To do this, we will say that the iteration has failed if $|V'_r| > (1-\varepsilon/2)|V'_{r-1}|$ after that iteration of while loop has terminated.

Within the while loop, there exist two for loops, one aiming to add a single element to the solution (starting in Line 6), and another aiming to add in multiple elements at once (starting in Line 14). We say that each iteration of those for loops is a step in our while loop, where $\lceil 1/\varepsilon \rceil$ of the steps are for the first for loop, and the other $O(\log k)$ steps are for the second while loop. Let $P_z$ be the upper bound on the probability that the while loop fails during step $z$. We examine the values of $P_z$ in the different cases.

- If $V'_{r-1}$ has at least $\varepsilon/2$ fraction of the elements with marginal gains lower than $\tau$, then at least $\varepsilon/2$ fraction of the elements will be removed from $V'_{r-1}$ anyway, and the iteration cannot fail (therefore $P_z = 0$ in these steps).

- Consider the first $\lceil 1/\varepsilon \rceil$ steps. Assume that less than $\varepsilon/2$ fraction of the elements in $V'_{r-1}$ have marginal gains lower than $\tau$. Then, when a random element is selected in one of the steps, it has probability of at most $\varepsilon/2$ of selecting an element which will result in the while loop terminating and failing. Therefore in this case, for $1 \leq z \leq \lceil 1/\varepsilon \rceil$, $P_z = \varepsilon/2$.

- Consider the remaining $O(\log k)$ steps. In these steps, we sample $t$ elements to form set $T$ and add them to $S$. We can see that the while loop will fail in one of these steps if the sampled set $T$ is such that $f(T|S) \leq (1-\varepsilon)\tau \cdot |T|$ (which causes the while loop to break), and less than $\varepsilon/2$ fraction of the elements in $V'_r$ has marginal gain towards $S \cup T$ of less than $\tau$ (which causes the while loop to fail).

  We can see that $f(T|S) \leq (1-\varepsilon)\tau \cdot |T|$ would only be true if at least $\varepsilon$ fraction of the elements in $T$ have marginal gain towards $S$ of less than $\tau$. Define a random variable $X$ which is equal to 1 if $f(x|S) < \tau$ for a random $x \in V'_r$ and 0 otherwise. We can see that the mean of $X$ is less than $\varepsilon/2$. Treat the process of sampling $T$ as sampling out $t$ individual elements from $V'_r$, and suppose we assign random variable $X_i$ for each of the random elements in $T$. The probability that more than $\varepsilon t$

of these elements have marginal gain towards $S$ of less than $\tau$ is given by

$$\Pr\left[\sum_{i=1}^{t} X_i \geq \varepsilon t\right] = \Pr\left[\sum_{i=1}^{t} X_i - \frac{\varepsilon t}{2} \geq \frac{\varepsilon t}{2}\right] \leq e^{-2(\varepsilon^2 t^2/4)/t} = e^{-\varepsilon^2 t/2}$$

by Hoeffding's Inequality (Theorem 3.3.2). This means that the while loop has a probability of less than $e^{-\varepsilon^2 t/2}$ to fail during this step. In this stage, $P_z = e^{-\varepsilon^2 t/2}$.

The probability that iteration $l$ of the while loop will *not* fail after all of these steps is therefore at least $\prod_z (1 - P_z)$, which is independent of $|V'_r|$. We can therefore say that $|V'_r| \leq (1 - \varepsilon/2)|V'_{r-1}|$ with constant probability. This means the number of elements in set $V'$ will be decaying exponentially. In the case that set $S$ never reaches $k$ elements, the while loop will continue running until $|V'| = 0$, which will happen in $O(\log |V|)$ iterations.

In iteration $r$, we would see that $|V'_r| \leq |V|(1 - \varepsilon/2)^r$, and so the number of submodular function calls required for that filtering stage is bounded by $|V|(1 - \varepsilon/2)^r$. After the filtering stage, the algorithm requires another $O(1/\varepsilon + \log k)$ submodular function calls. Overall, the number of submodular function calls required for THRESHOLD-SAMPLING would be bounded by

$$\sum_{r=1}^{C \log |V|} \left[|V|(1 - \varepsilon/2)^r + \frac{1}{\varepsilon} + \log k\right] \leq \frac{2|V|}{\varepsilon} + \frac{C \log |V|}{\varepsilon} + C \log |V| \log k$$

meaning that the work required for the algorithm is $O(|V|/\varepsilon)$. An interesting point to note here is the runtime is dominated by the filtering stage.

To prove the second claim, we can examine the average marginal gain in each cases where elements are added to the set $S$. Suppose we are in iteration $r$ of the while loop. Suppose we let $T_{r,j}$ be the $j$th batch of element(s) which were added into the solution during iteration $r$ of the while loop, and let $S_{r,j}$ be the solution right before adding $T_{r,j}$ into the solution set. For iteration $r$ of the while loop, there are three ways it can terminate.

- By the break in Line 9. Here, we see that $|T_{r,j}| = 1$ for all $j$. The elements here could only be added in Line 11, and by the condition of the if statement we can guarantee that $f(T_{r,j}|S_{r,j}) \geq (1 - \varepsilon)\tau$.

- By the break in Line 23. Suppose are about to add $T_{r,j}$ into our solution set in Line 21. We can see that in order for the while loop to have reached this point, elements must only have been added in Line 11 and Line 18, otherwise the loop would have been terminated already. This means that for previous $j-1$ batches of elements added, the average marginal gain must be higher than $(1-\varepsilon)\tau$. Also, we can see that from how $t$ is constructed, we see that $\left|\bigcup_{l=1}^{j-1} T_{r,l}\right| = (1+\varepsilon)^i$ and that $\left|\bigcup_{l=1}^{j} T_{r,l}\right| \leq (1+\varepsilon)^{i+1} = (1+\varepsilon)\left|\bigcup_{l=1}^{j-1} T_{r,l}\right|$. We can therefore show that

$$\begin{aligned}
f\left(\bigcup_{l=1}^{j} T_{r,l}\middle| S_{r,1}\right) &\geq f\left(\bigcup_{l=1}^{j-1} T_{r,l}\middle| S_{r,1}\right) && \text{by submodularity} \\
&\geq (1-\varepsilon)\tau \cdot \left|\bigcup_{l=1}^{j-1} T_{r,l}\right| && \text{by condition of added elements} \\
&\geq \frac{(1-\varepsilon)\tau}{(1+\varepsilon)} \cdot \left|\bigcup_{l=1}^{j-1} T_{r,l}\right| \\
&\geq (1-2\varepsilon)\tau \cdot \left|\bigcup_{l=1}^{j} T_{r,l}\right| && \text{by approximation}
\end{aligned}$$

  which means the average marginal gain for elements added in iteration $r$ of the while loop exceeds $(1-2\varepsilon)\tau$ as required.

- The while loop does not reach any break conditions and terminates after finishing all the for loops. For the algorithm to not reach any break conditions, it must mean that $f(T_{r,j}|S_{r,j}) \geq (1-\varepsilon)\tau$ for all $j$, which matches the required condition.

We see that in any iteration of the while loop the average marginal gains of the elements are as required.

To prove the third claim, we can see that if $|S| < k$, then $|V'| = 0$. This means that no elements remain from $V'$, which would be due to the fact that all the elements from it is either added to $S$ or filtered out in one of the iterations. If the elements are filtered out, then it must mean that in some iteration $r$, the marginal gain of the element towards $S_{r,1}$ is lower than $\tau$ as in the condition from FILTER subroutine. They must therefore also have marginal gain of less than $\tau$ towards $S$ as well due to submodularity. □

THRESHOLD-SAMPLING alone is not enough to obtain a final solution, as we do not know what

$\tau$ should be. Ideally, we would let $\tau = \texttt{OPT}/2k$, where $\texttt{OPT}$ is the true optimal solution. Since we do not know $\texttt{OPT}$, we can instead run multiple copies of THRESHOLD-SAMPLING for different estimates of $\texttt{OPT}$, and take the best solution obtained.

### 3.3.2 The EXHAUSTIVE-MAXIMISATION Algorithm

In the paper by Fahrbach, Mirrokni and Zadimoghaddam [FMZ18b], a similar idea is used in order to estimate the optimal solution. In the EXHAUSTIVE-MAXIMISATION function in Algorithm 8, they have let $\Delta^* = \max_{v \in V} f(\{v\})$. Then, we can observe that $\Delta^* \leq \texttt{OPT} \leq k\Delta^*$, where the first inequality is due to the fact that $\Delta^*$ is a feasible value for the maximisation, and the second inequality is due to the property of diminishing returns of submodular functions. Then, solutions are found in parallel for different values of $\tau = (1+\varepsilon)^i \Delta^*/k$ for integers $i$ such that $\Delta^* \leq \tau \leq k\Delta^*$. These solutions are found by running THRESHOLD-SAMPLING-FAHRBACH (Algorithm 9) for decreasing threshold values in order to achieve the guarantees provided.

THRESHOLD-SAMPLING-FAHRBACH shares similar techniques to THRESHOLD-SAMPLING from earlier. It first filters out elements whose marginal gain is too low for the solution set. Then, it estimates how many elements it can sample out of unselected elements such that the sampled set will give some marginal gain to the solution. This is done using an estimator function REDUCE-MEAN which is shown in Algorithm 10. Finally, the algorithm samples out that many elements and adds it to the solution set. This is repeated until enough elements are added to the solution or the algorithm runs out of unfiltered elements.

Some key theorems about Algorithms 8, 9 and 10 are summarised below. All of their proofs are omitted here, but can be found in the original paper.

**Theorem 3.3.3.** *Let $\mu$ be the mean of the Bernoulli distribution $\mathcal{D}$. Then, with probability of at least $1-\delta$,*

1. *If* REDUCED-MEAN *returns* `True`, then $\mu \leq 1 - \varepsilon$.

2. If REDUCED-MEAN returns `False`, then $\mu \geq 1 - 2\varepsilon$.

**Theorem 3.3.4.** *The algorithm* THRESHOLD-SAMPLING-FAHRBACH *returns a set $S \subseteq V$ such that, with probabilities of at least $1-\delta$,*

1. *The expected number of queries to the submodular function is $O(|V|/\varepsilon)$,*
2. *The expected average marginal gain for each element $\mathbb{E}[f(S)/|S|]$ is greater than $(1-\varepsilon)\tau$, and*
3. *If $|S| < k$, then $f(x|S) < \tau$ for all $x \in V$.*

**Theorem 3.3.5.** *Given a monotone submodular function $f$,* EXHAUSTIVE-MAXIMISATION *returns a set $S \subseteq V$ with $|S| \leq k$ in $O(\log(n/\delta)/\varepsilon^2)$ adaptive rounds such that with probability at least $1-\delta$, the algorithm makes $O(n\log(k)/\varepsilon^3)$ submodular function calls in expectation, and $\mathbb{E}[f(S)] \geq (1-1/e-\varepsilon)\texttt{OPT}$.*

**Algorithm 8** EXHAUSTIVE-MAXIMISATION Algorithm as presented in [FMZ18b]

**function** EXHAUSTIVE-MAXIMISATION($V, f, k, \varepsilon, \delta$)
  $\Delta^* \leftarrow \max_{v \in V} f(\{v\})$
  $r \leftarrow \lceil 2\log(k)/\varepsilon \rceil$
  $m \leftarrow \lceil \log(4)/\varepsilon \rceil$
  $\hat{\delta} \leftarrow \delta/(r(m+1))$
  $\mathcal{R} \leftarrow \emptyset$
  **for** $i = 0$ to $r$ in parallel **do**
    $\tau \leftarrow (1+\varepsilon)^i \Delta^*/k$
    $S \leftarrow \emptyset$
    **for** $j = 0$ to $m$ **do**
      define a function $f_S(T) = f(S \cup T) - f(S)$
      $T \leftarrow$ THRESHOLD-SAMPLING-FAHRBACH$(V, f_S, k - |S|, (1-\varepsilon)^j\tau, \varepsilon, \hat{\delta})$
      $S \leftarrow S \cup T$
      **if** $|S| = k$ **then**
        **break**
    add $S$ to $\mathcal{R}$
  **return** $\arg\max_{S \in \mathcal{R}} f(S)$

## 3.4 A Practical Parallel Submodular Maximisation Algorithm

In this section, we present our own submodular maximisation algorithm under cardinality constraints. We first present the shortcomings of existing algorithms, then we propose an algorithm that fixes these issues. Finally, we theoretically show the approximation bounds and running time of our algorithm.

**Algorithm 9** THRESHOLD-SAMPLING algorithm from [FMZ18b]

**function** THRESHOLD-SAMPLING-FAHRBACH($V$, $f$, $k$, $\tau$, $\varepsilon$, $\delta$)
  $\hat{\varepsilon} \leftarrow \varepsilon/3$
  $r \leftarrow \lceil \log_{(1-\hat{\varepsilon})^{-1}}(2|V|/\delta) \rceil$
  $m \leftarrow \lceil \log(k)/\hat{\varepsilon} \rceil$
  $\hat{\delta} \leftarrow \delta/(2r(m+1))$
  $S \leftarrow \emptyset$
  $A \leftarrow V$
  **for** $r$ rounds **do**
    $A \leftarrow$ FILTER$(A, f, S, \tau)$ ▷ Algorithm 7
    **if** $|A| = 0$ **then break**
    **for** $i = 0$ to $m$ **do**
      $t \leftarrow \min\{\lfloor (1+\hat{\varepsilon})^i \rfloor, |A|\}$
      define $\mathcal{D}_t$ as a distribution over $\mathbb{1}[f(x|S \cup T) \geq \tau]$ for $T \sim \mathcal{U}(A, t-1)$ and $x \sim A \setminus T$
      **if** REDUCED-MEAN$(\mathcal{D}_t, \hat{\varepsilon}, \hat{\delta})$ **then**
        **break**
    sample $T \sim \mathcal{U}(A, \min\{t, k - |S|\})$
    $S \leftarrow S \cup T$
    **if** $|S| = k$ **then break**
  **return** $S$

**Algorithm 10** An estimator to Bernoulli distribution

**function** REDUCED-MEAN($\mathcal{D}, \varepsilon, \delta$)
  $m \leftarrow 16\lceil \log(2/\delta)/\varepsilon^2 \rceil$
  sample $X_1, X_2, \ldots, X_m \sim \mathcal{D}$ in parallel
  $\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} X_i$
  **if** $\mu \leq 1 - 1.5\varepsilon$ **then**
    **return** `True`
  **else**
    **return** `False`

### 3.4.1 A Problem With Parallel Algorithms Discussed

A big issue which prevents some of the theoretically efficient parallel algorithm from being efficient in real life is the amount of actual parallelism which is required. In many of these algorithms discussed, we find that while the theoretical adaptivity is low, the number of actual parallel function calls which would be required to take advantage of the low adaptivity is simply too large for them to be fully parallel when run on physical machines. This means that running multiple copies of the same subroutine can be done less efficiently, and should be avoided if possible.

We also found that while a lot of the existing parallel algorithms have low adaptivity or even query complexity, they have a large constant hidden in its asymptotic running time. This means that the algorithm would only be practical if we ran the test for a large enough set $V$ (for exampled, for algorithms presented by Fahrbach, Mirrokni and Zadimoghaddam, the size is in the order of $10^8$). This means that the algorithm provides little advantage in the smaller samples, or in data which we are likely to encounter.

### 3.4.2 The two versions of THRESHOLD-SAMPLING

Algorithms 6 and 9 are both subroutines which attempt to add elements to the solution so that each added element has a marginal gain larger than some threshold amount. Both algorithms also try to add multiple elements at a time in order to be more efficient. Theorem 3.3.1 and Theorem 3.3.4 both highlight the differences in the guarantees obtained from Algorithms 6 and 9 respectively.

Another major difference between the two subroutines is their probabilistic nature. Algorithm 6 is only probabilistic in the sense that its runtime may vary simply due to chance, while the returned set will always have some amount of marginal gain. On the other hand, Algorithm 9 also includes a failure rate, which means that there is a probability that the answer returned will not be sufficiently accurate, although the number of query calls is asymptotically smaller compared to Algorithm 6.

In practice, we find that Algorithm 6 will run faster since it does not depend on the failure rate (which can also be adjusted). Also, efficiently implementing the required distribution function (Line 13 of Algorithm 9) is not trivial, which can cause large performance setbacks.

### 3.4.3 Our Proposed Algorithm

Using the THRESHOLD-SAMPLING subroutine (Algorithm 6), we present COOL-SUBMOD, which is a parallel submodular maximisation algorithm that is more practical than previously proposed algorithms. We outline our ideas in Algorithm 11.

In our algorithm, we use a greedy approach which iteratively calls on THRESHOLD-SAMPLING to add a set of new elements whose average marginal gain is higher than some fixed amount. However, instead of calling THRESHOLD-SMAPLING on the entire set $V$, we only call it on a selected set of elements whose marginal gain might be high enough for our threshold. We can do so by first grouping all of the elements into different bins (labelled $B_i$) according to their marginal gain towards the set $S$ using the REDISTRIBUTE subroutine. In iteration $t$, we call THRESHOLD-SAMPLING on elements in $B_t$. After THRESHOLD-SAMPLING is called, the unselected element from the bin $B_t$ is redistributed into the lower bins according to their marginal gain using the REDISTRIBUTE subroutine once more.

Finally, we terminate the algorithm if either the cardinality constraint is met, or all of the remaining elements in the pool have too low of a marginal gain. When the marginal gains of any additional elements are too low, we can terminate early since extra elements added to our set would not be worth the extra computation time, and so we can simply add random elements to reach the maximum cardinality instead.

Note that since we do not require an estimate of `OPT`, we do not need to run multiple copies of this algorithm in parallel. This further increases the practicality of our algorithm as we do not require as many tasks to be done in parallel.

### 3.4.4 Theoretical Analysis of COOL-SUBMOD

We will now analyse the performance of our proposed algorithm. For simplicity, we will define variables that represent the state of the algorithm in the while loop at Line 7. For iteration $j$ of the while loop (where $j$ starts at 1), define $\Delta_j = (1-\hat{\varepsilon})^{j+1}\Delta^*$ to be the threshold value at that iteration (as defined in the THRESHOLD-SAMPLING call). Also let $T_j$ be the set returned from THRESHOLD-SAMPLING in iteration $j$, and $n_i$ be the size of set $S$ at the end of that iteration. Assume the elements are added into $S$

**Algorithm 11** Our proposed submodular maximisation algorithm

**function** COOL-SUBMOD($V, f, k, \varepsilon$)
  $\hat{\varepsilon} = \varepsilon/(2 - 1/e)$ ▷ set smaller error term
  $\Delta^* \leftarrow \max_{v \in V} f(\{v\})$
  $S \leftarrow \emptyset$ ▷ set of solution
  REDISTRIBUTE($V, f, S, \Delta^*, \hat{\varepsilon}$)
  $t \leftarrow 0$ ▷ keeps current bin to check
  **while** $|S| < k$ and $(k - |S|) \cdot (1 - \hat{\varepsilon})^{t+1}\Delta^* > \hat{\varepsilon} \cdot f(S)$ **do**
    **if** $B_t$ exists **then**
      define $f_S(T) = f(S \cup T) - f(S)$
      $T \leftarrow$ THRESHOLD-SAMPLING($B_t, f_S, k - |S|, (1 - \hat{\varepsilon})^{t+1}\Delta^*, \hat{\varepsilon}/2$) ▷ Algorithm 6
      $S \leftarrow S \cup T$
      $B_t \leftarrow B_t \setminus T$
      REDISTRIBUTE($B_t, f, S, \Delta^*, \hat{\varepsilon}$)
    $t \leftarrow t + 1$
  **if** $|S| < k$ **then**
    add $k - |S|$ random elements from $V \setminus S$ to $S$
  **return** $S$

**Algorithm 12** The redistribution subroutine

**function** REDISTRIBUTE($A, f, S, \Delta^*, \varepsilon$)
  **for** $v \in A$ in parallel **do**
    $i \leftarrow \lceil \log_{1-\varepsilon}( f(S \cup \{v\})/\Delta^* ) \rceil$
    **if** $B_i$ does not exist **then**
      $B_i \leftarrow \{v\}$ ▷ create a new bin globally
    **else**
      $B_i \leftarrow B_i \cup \{v\}$

sequentially, and that the elements in $S$ are kept in order according to when it was added into the solution set. For $1 \leq i \leq |S|$, define $v_i$ to be the $i$th element in $S$, and let $S_i = \{v_1, v_2, \ldots, v_i\}$. In other words, we can say that $S_{n_j} = S_{n_{j-1}} \cup T_j$, and $S_0 = \emptyset$. Additionally, define a small error term $\hat{\varepsilon} = \frac{\varepsilon}{2-1/e}$ as set in Line 2 of the algorithm.

**Approximation Accuracy of COOL-SUBMOD**

Ultimately, we would like to claim that our algorithm produces an approximation guarantee which is arbitrarily close to the approximation guarantee for the greedy algorithm. We begin with a claim about the output from calling THRESHOLD-SAMPLING in Line 10.

**Lemma 3.4.1.** *Suppose the while loop in Line 7 has run for $t$ iterations. In this iteration, the* THRESHOLD-SAMPLING *call returns a set $T_t$ such that*

1. *The marginal gain of elements in $T_t$ is given by $f(T_t|S_{n_{t-1}}) \geq |T_t| \cdot (1-\hat{\varepsilon})\Delta_t$.*
2. *For all elements $x \in V \setminus S_{n_t}$, we have $f(x|S_{n_t}) < \Delta_t$.*

*Proof.* We can see that this theorem is a consequence of Theorem 3.3.1 for the set of parameters we have defined. The first part of the lemma comes from the fact that the threshold for added items in this case is set at $\Delta_t = (1-\hat{\varepsilon})^{t+1}\Delta^*$, and that we have set the error term to $\hat{\varepsilon}/2$.

For the second part of the lemma, we can see that all the elements in $V \setminus S_{n_t}$ are either in $B_t$ at the beginning of the while loop or are not. If they are in $B_t$ at the beginning of the while loop, they are not added into $S$ because they were removed from the pool of elements in the filtering stage of THRESHOLD-SAMPLING, meaning their marginal gain towards $S$ is lower than $\Delta_t$. On the other hand, if the element is not in $B_t$, then they must be in a lower bin. This would mean that their marginal gain towards the solution from an earlier iteration is lower than $\Delta_t$, and so their marginal gain towards the current solution $S$ must be lower than $\Delta_t$ as well by submodularity. □

Using this result, we will prove the performance of entire algorithm. We will first start by proving the performance of a modified version of the algorithm, where we only terminate out of the while loop in

the case that $|S| = k$, and not under the condition that $(k - |S|) \cdot \Delta_t \leq \hat{\varepsilon} \cdot f(S)$. After that, we will show that the objective value when we do terminate under the condition $(k - |S|) \cdot \Delta_t \leq \hat{\varepsilon} \cdot f(S)$ decreases only by a constant fraction.

We will first prove the approximation guarantee for the algorithm without the extra termination conditions. The following proof is inspired by [FMZ18b], and shares similarities to the proof for the correctness of the greedy algorithm (Theorem 3.2.2).

Define a variant of our submodular function called $\hat{f}$, which averages the marginal gain in each iteration of the while loop. In other words, we define the function such that for any iteration $t$, for any $n_t \leq i < n_{t+1}$ we have $\hat{f}(v_i|S_{i-1}) = f(T_t|S_{n_t-1})/|T_t|$. We can see that all the elements which are added to $S$ during the same iteration of the while loop will have the same marginal gain towards $\hat{f}$, and this is equal to the average marginal gain of elements from $T_t$ towards the original submodular function $f$. We can also see that for the final solution set $S$, we would have $f(S) = \hat{f}(S)$. Furthermore, since the average marginal gain is found only on elements from the same while loop, for any iteration $t$, we find that $f(S_{n_t}) = \hat{f}(S_{n_t})$. We can prove an invariance for $\hat{f}$.

**Lemma 3.4.2.** *For $1 \leq i \leq |S|$, we have*

$$\hat{f}(v_i|S_{i-1}) \geq \frac{(1-\hat{\varepsilon})^2}{k}(\texttt{OPT} - \hat{f}(S_{i-1})).$$

*Proof.* We will prove this by induction. We can see that since there will be at least one element in $B_0$ (which will have a marginal gain to the empty set of $\Delta^*$), it will get added to the set $S$ in the first iteration of the while loop. This means that at least one element will be added in the first iteration of the while loop, or for

when $t = 0$. For this element, we can see that

$$\begin{aligned}
\hat{f}(S_1) &= \hat{f}(\{v_1\}) \\
&= \frac{f(\{v_1\})}{|T_0|} && \text{by definition of } \hat{f} \\
&\geq \frac{|T_0| \cdot (1-\hat{\varepsilon})^2 \cdot \Delta^*}{|T_0|} && \text{by Lemma 3.4.1} \\
&\geq (1-\hat{\varepsilon})^2 \cdot \frac{\texttt{OPT}}{k} && \text{by submodularity} \\
&\geq \frac{(1-\hat{\varepsilon})^2}{k}(\texttt{OPT} - \hat{f}(S_0))
\end{aligned}$$

which proves the base case.

Now, we will prove the inductive step. Assume that $\hat{f}(v_{i-1}|S_{i-2}) \geq \frac{(1-\hat{\varepsilon})^2}{k}(\texttt{OPT} - \hat{f}(S_{i-2}))$ is true. There are two possibilities, either $v_i$ and $v_{i-1}$ are both added during the same iteration of the while loop, or they are added during a different iteration. We will consider both of these cases separately.

First, consider the case when $v_i$ and $v_{i-1}$ is added in separate iterations of the while loop. This means that $v_i$ was the first element added in its iteration of the while loop. Suppose $v_i$ was added in iteration $t$. This would mean that $i = n_{t-1} + 1$. Assuming that $f(S^*) = \texttt{OPT}$, we can see that

$$\begin{aligned}
\texttt{OPT} = f(S^*) &\leq f(S^* \cup S_{i-1}) && \text{by submodularity} \\
&\leq f(S_{i-1}) + \sum_{x \in S^*} f(x|S_{i-1}) && \text{by submodularity} \\
&\leq f(S_{i-1}) + \sum_{x \in S^*} \Delta_{t-1} && \text{by Lemma 3.4.1} \\
&\leq f(S_{i-1}) + k \cdot \Delta_{t-1} \\
&\leq f(S_{i-1}) + \frac{k}{(1-\hat{\varepsilon})} \cdot \Delta_t \\
&\leq f(S_{i-1}) + \frac{k}{(1-\hat{\varepsilon})^2} \cdot \frac{f(T_t|S_{i-1})}{|T_t|} && \text{by Lemma 3.4.1} \\
&= f(S_{i-1}) + \frac{k}{(1-\hat{\varepsilon})^2} \cdot \hat{f}(v_i|S_{i-1}) && \text{by definition of } \hat{f}.
\end{aligned}$$

Also, since $i - 1 = n_{t-1}$, it is the case that $f(S_{i-1}) = \hat{f}(S_{i-1})$ since $S_{i-1}$ has no partial contributions from any while loop. Rearranging the expression above, we can arrive at the conclusion that $\hat{f}(v_i|S_{i-1}) \geq \frac{(1-\hat{\varepsilon})^2}{k}(\texttt{OPT} - \hat{f}(S_{i-1}))$.

Now consider the other case that $v_i$ and $v_{i+1}$ are both added during the same iteration of the while loop. Since the two elements are added in the same iteration of the while loop, their marginal gain towards $\hat{f}$ are the same by definition. This means that

$$\begin{aligned}
\hat{f}(v_i|S_{i-1}) &= \hat{f}(v_{i-1}|S_{i-2}) && \text{by definition of } \hat{f} \\
&= \frac{(1-\hat{\varepsilon})^2}{k}(\texttt{OPT} - \hat{f}(S_{i-2})) && \text{by the inductive hypothesis} \\
&\geq \frac{(1-\hat{\varepsilon})^2}{k}(\texttt{OPT} - \hat{f}(S_{i-1})) && \text{by submodularity}
\end{aligned}$$

which proves that the statement is true in this case as well. This means that by induction, $\hat{f}(v_i|S_{i-1}) \geq \frac{(1-\hat{\varepsilon})^2}{k}(\texttt{OPT} - \hat{f}(S_{i-1}))$ is true for all $1 \leq i \leq |S|$. □

Using the invariance for $\hat{f}$, we can prove the following result for the simplified version of the algorithm.

**Theorem 3.4.1.** *Consider a variant of Algorithm 11 where the while loop in Line 7 only terminates when $|S| = k$. At termination, the algorithm will return a set $S$ such that $f(S) \geq (1 - 1/e - \hat{\varepsilon})\texttt{OPT}$.*

*Proof.* Let $\hat{\delta}_i = \texttt{OPT} - \hat{f}(v_i|S_{i-1})$. We can see that

$$\begin{aligned}
\hat{\delta}_i - \hat{\delta}_{i+1} &= \hat{f}(v_{i+1}|S_i) - \hat{f}(v_i|S_{i-1}) \\
&= \hat{f}(S_{i+1}) - f(S_i) \\
&= \hat{f}(v_{i+1}|S_i) \\
&\geq \frac{(1-\hat{\varepsilon})^2}{k} \cdot \hat{\delta}_i && \text{by Lemma 3.4.2}
\end{aligned}$$

and so $\hat{\delta}_{i+1} \leq \left(1 - \frac{(1-\hat{\varepsilon})^2}{k}\right)\hat{\delta}_i$. By continuing the same pattern, we can see that

$$\begin{aligned}
\texttt{OPT} - \hat{f}(S) &= \hat{\delta}_k \\
&\leq \left(1 - \frac{(1-\hat{\varepsilon})^2}{k}\right)^k \cdot \hat{\delta}_0 \\
&= \left(1 - \frac{(1-\hat{\varepsilon})^2}{k}\right)^k \texttt{OPT}.
\end{aligned}$$

From algebra, we can show that

$$\left(1-\frac{(1-\hat{\varepsilon})^2}{k}\right)^k = \left[\left(1-\frac{(1-\hat{\varepsilon})^2}{k}\right)^{\frac{k}{(1-\hat{\varepsilon})^2}}\right]^{(1-\hat{\varepsilon})^2} \leq e^{-(1-\hat{\varepsilon})^2} \leq \frac{1}{e}+\hat{\varepsilon}$$

which means that $\hat{f}(S) \geq (1-1/e-\hat{\varepsilon})\texttt{OPT}$. Finally, we know that $\hat{f}(S) = f(S)$ by the way $\hat{f}$ was constructed, which means that $f(S) \geq (1-1/e-\hat{\varepsilon})\texttt{OPT}$ which proves our theorem. □

Finally, we will use the theorem above to give an approximation guarantee for COOL-SUBMOD which has the termination condition as stated in the algorithm.

**Theorem 3.4.2.** *For a monotone submodular function $f$ and cardinality constraint $k$,* COOL-SUBMOD *produces a solution $S$ such that $f(S) \geq (1-1/e-\varepsilon)\ \texttt{OPT}$.*

*Proof.* Suppose COOL-SUBMOD terminates at iteration $t$ of the while loop with a solution $S$. We can see that if COOL-SUBMOD were allowed to run beyond the termination condition, the extra marginal gain it may get is no larger than $(k-|S|)\cdot\Delta_t$, which by the termination condition is no larger than $\hat{\varepsilon}f(S)$. Let $S'$ be the solution one would obtain if they ran COOL-SUBMOD without the termination condition. We can then see that $f(S')-f(S) \leq \hat{\varepsilon}f(S)$, and so $f(S) \geq f(S')/(1+\hat{\varepsilon})$.

By Theorem 3.4.1, we know that $f(S') \geq (1-1/e-\hat{\varepsilon})\texttt{OPT}$. This means that

$$\begin{aligned}
f(S) &\geq \left(\frac{1-\frac{1}{e}-\hat{\varepsilon}}{1+\hat{\varepsilon}}\right)\texttt{OPT} \\
&\geq \left(1-\frac{1}{e}-\hat{\varepsilon}\right)(1-\hat{\varepsilon})\texttt{OPT} \\
&= \left[1-\frac{1}{e}-\left(2-\frac{1}{e}\right)\hat{\varepsilon}+\hat{\varepsilon}^2\right]\texttt{OPT} \\
&\geq \left[1-\frac{1}{e}-\left(2-\frac{1}{e}\right)\hat{\varepsilon}\right]\texttt{OPT} \\
&\geq \left(1-\frac{1}{e}-\varepsilon\right)\texttt{OPT} \qquad\qquad \text{since } \varepsilon = \left(2-\frac{1}{e}\right)\hat{\varepsilon}
\end{aligned}$$

which proves our theorem. □

**Runtime of COOL-SUBMOD**

**Theorem 3.4.3.** *The while loop in Line 7 of* COOL-SUBMOD *runs at most* $\log_{1/(1-\hat{\varepsilon})}(k/\hat{\varepsilon})$ *times.*

*Proof.* Suppose the while loop has repeated for $t$ iterations where $t \geq \log_{1/(1-\hat{\varepsilon})}(k/\hat{\varepsilon})$. In the $t$th iteration the value of the threshold is $\Delta_t = (1+\hat{\varepsilon})^t \Delta^*$. We can therefore see that

$$\begin{aligned}
\Delta_t &= (1+\hat{\varepsilon})^t \Delta^* \\
&\leq \frac{\hat{\varepsilon}}{k} \cdot \Delta^* && \text{by definition of } k \\
&\leq \frac{\hat{\varepsilon}}{k} \cdot f(S) && \text{by submodularity} \\
&\leq \frac{\hat{\varepsilon}}{k-|S|} \cdot f(S)
\end{aligned}$$

which means that $(k-|S|) \cdot \Delta_t \leq \hat{\varepsilon} \cdot f(S)$. Under this situation the while loop will terminate. We can therefore conclude that the while loop will repeat no more than $\log_{1/(1-\hat{\varepsilon})}(k/\hat{\varepsilon})$ times. □

We can see that since $\log(1+\hat{\varepsilon})$ is approximately equal to $1+\hat{\varepsilon}$ for small $\hat{\varepsilon}$, by Theorem 3.4.3, we can see that our while loop runs $O\big(\frac{\log(k/\varepsilon)}{\varepsilon}\big)$ times.

The amount of work done within a while loop itself will depend on the size of the bin $B_t$ at the beginning of the while loop. However, since the bins are constructed at runtime, it is tricker to bound the work required to process each bin. We can see that if we define our submodular function as

$$f(S) = \sum_{i=1}^{|S|} (1-\hat{\varepsilon})^i$$

then when we get to iteration $t$ of the while loop the size of $B_t$ will always be $|V|-t$ since in each iteration only one element will be added to $S$ at a time, and it will move all of the remaining elements into bin $B_{t+1}$. In practice, however, this worst case scenario is unlikely to occur.

## 3.5 Tests of Our Algorithm

In this section, we describe the setup for benchmarking our algorithm.

### 3.5.1 Implementation

All of the algorithms were implemented in Julia. There were a couple of reasons why we chose to implement the algorithms in Julia. Firstly, Julia is fast. Its speed is comparable to languages like C and C++, which is better for our tasks which require a lot of computation. Secondly, it has got more ready mathematical tools, such as its `LinearAlgebra` package which can calculate matrices determinant, a function that is needed in some submodular functions we are testing. Julia is also "friendlier" than programming languages like C or C++, and are more popular amongst data scientists who prefers to avoid low-level languages. We therefore decided that functions implemented in Julia would potentially have the most use to others who may want to work on similar problems.

For parallelisation of the algorithms, we are using the `Threads` package on Julia. While we do acknowledge that the feature is marked as experimental (as of Julia version 1.4.1), we find that it is stable enough that we do not face major issues that makes it unsuitable for our task, and that there is enough online supports for `Threads` package from other Julia users. However, during the implementation process, we also found that multithreading in Julia does not handle task scheduling as well as we expected, and we avoided implementing nested parallelised loops, instead changing one of the loops to be sequentially executed instead.

### 3.5.2 Testing Setup

All of the algorithms are run on a machine with four threads and 8GB of RAM on Digital Ocean. We recorded the wall clock time to perform the maximisation using the various algorithms, and also the objective values that the algorithm gives. Furthermore, we also counted the number of total submodular function queries each algorithm makes, which will show the amount of work the algorithm has to do. A large number of query calls would indicate that the algorithm requires a lot of computation, which would be independent of the number of threads.

For each objective value, we have varied the value of $k$ to be values between 2 to 1024. We cap each testing at two hours, and we terminate the computation for that test if the runtime exceeds this.

### 3.5.3 Algorithms Benchmarked

We will benchmark COOL-SUBMOD against the following algorithms. All of the algorithms listed here have been implemented and tested in Julia.

- RANDOM, which is the random algorithm. In this case, we keep picking random elements and adding them to the solution set until the cardinality constraint is met.
- GREEDY, which is the parallelised greedy algorithm, which is a variant to Algorithm 5 however the search for the next element to add to the solution is done in parallel.
- The SIEVE-STREAMING algorithm presented by Badanidiyuru et al [BMKK14]. While this algorithm is designed for the streaming setting (which is different from the cases we are dealing with here), its performance is still an interesting comparison.
- A modification of SIEVE-STREAMING-PLUS algorithm presented by Kazemi et al [KMZ$^+$19]. This is the paper which presents the THRESHOLD-SAMPLING subroutine (Algorithm 3.3.1). In the actual algorithm, the data is processed in batches. In our tests, however, we will allow the algorithm to see all of the data at once (i.e. processing the data in a single batch) since we are not testing streaming algorithms in this case.
- FAHRBACH, which is the algorithms presented by Fahrbach, Mirrokni and Zadimoghaddam [FMZ18b]. We have implemented the THRESHOLD-SAMPLING-FAHRBACH (Algorithm 9) and EXHAUSTIVE-MAXIMISATION algorithm (Algorithm 8). To speed up this algorithm, we will modify the REDUCE-MEAN function (Algorithm 10) from the paper so that it is able to terminate earlier in practice while returning the correct answer (at the cost of higher adaptivity). We call our variation of the function ADJUSTED-REDUCE-MEAN, which is introduced in Appendix A.1. We also used the BINARY-SEARCH-MAXIMISATION subroutine which is presented in the original paper in order to reduce the number of copies of EXHAUSTIVE-MAXIMISATION instances that needs to be run.

Additionally, we will compare some variants of our algorithm, which include the following.

- FAHRBACH-MODIFIED, which is a modification of FAHRBACH described above. In Algorithm 8, rather than calling the THRESHOLD-SAMPLING-FAHRBACH subroutine, it calls the THRESHOLD-SAMPLING subroutine (Algorithm 6) with appropriate parameters to keep the same approximation guarantee.

- COOL-SUBMOD, which is our proposed algorithm as presented in Algorithm 11.

For FAHRBACH, FAHRBACH-MODIFIED, SIEVE-STREAMING, SIEVE-STREAMING-PLUS and COOL-SUBMOD algorithms, we test their performance when setting $\varepsilon = 0.1$ and $\varepsilon = 0.01$. We kept all failure rates at $\delta = 0.1$.

### 3.5.4 Dataset

We perform submodular maximisation on different objectives and dataset listed below.

- Clustering objective on 10,000 uniformly-distributed random points. All of the points are two-dimensional and their components are distributed randomly between $[0, 1]$. For this case, we have let $e_0$ be the zero vector.

- Active set selection on 10,000 uniformly-distributed random points. All of the points are two-dimensional and their components are distributed randomly between $[0, 1]$. We have set the regularisation factor be equal to $1$.

- Movie recommendation objective using Movielens-1M dataset [HK15]. This dataset contains a set of 6000 users and 4000 movies, and one million ratings of the movies given by the users. Note that not all movies are rated by all of the users. Using the given ratings in the dataset, we convert each movies and users in the dataset into vectors using collaborative filtering, such that the dot product between a user vector and a movie vector is an estimate of the rating the particular user gives to the particular movie. This technique also allows for an estimate for the ratings that a user would have given to a movie which they have not actually rated. We have used the objective function as shown earlier, with $\alpha = 0.85$.

## 3.6 Results

We present the results which contain the objective values obtained, the required amount of computation time and the number of function calls.

The data on obtained objective values is highlighted in Figures 3.1 and 3.2. We find that streaming algorithms will do the worst, which is expected as their approximation guarantees are lower. We also find that COOL-SUBMOD performs almost as good as GREEDY and as good as the other parallel algorithms. All of the parallel algorithms give results at least 95% as good as random. We can also see the effects of the $\varepsilon$ dependency, that when the $\varepsilon$ is lower we also get a higher objective value.

The data on the runtime is presented in Figure 3.3. We can see that the amount of time for greedy in lower values of $k$ is actually less than that of most algorithms, however its runtime grows very quickly for larger $k$. We also find that for the two parallel algorithms the time required are both very high. On the other hand, COOL-SUBMOD is able to perform the computation with computation time lower by around an order of magnitude. We find that COOL-SUBMOD is sometimes faster than streaming algorithms as well. The same trend can be seen for the number of required query calls, which is as presented in Figure 3.4.

## 3.7 Conclusion

In this chapter, we have introduced COOL-SUBMOD, which is a practical algorithm for parallel submodular maximisation under cardinality constraints. We find that our algorithm performs as well as thee greedy algorithm, with much lower execution time and query calls required compared to the greedy algorithm and the other existing parallel algorithms.

In the future, we would like to further measure the performance of our algorithm against other submodular functions. We are also interested in seeing how well our algorithm will scale for larger values of $k$ than we have tested, and the computation time for cases which are longer than that we have tested. Furthermore, we would also like to test how our algorithm scales when we use a machine with more threads.

From a theoretical point of view, we would also like to also try to provide a better algorithmic bound for the algorithm. Specifically, we would like to find a way to bound the size of the bins $B_t$ in each

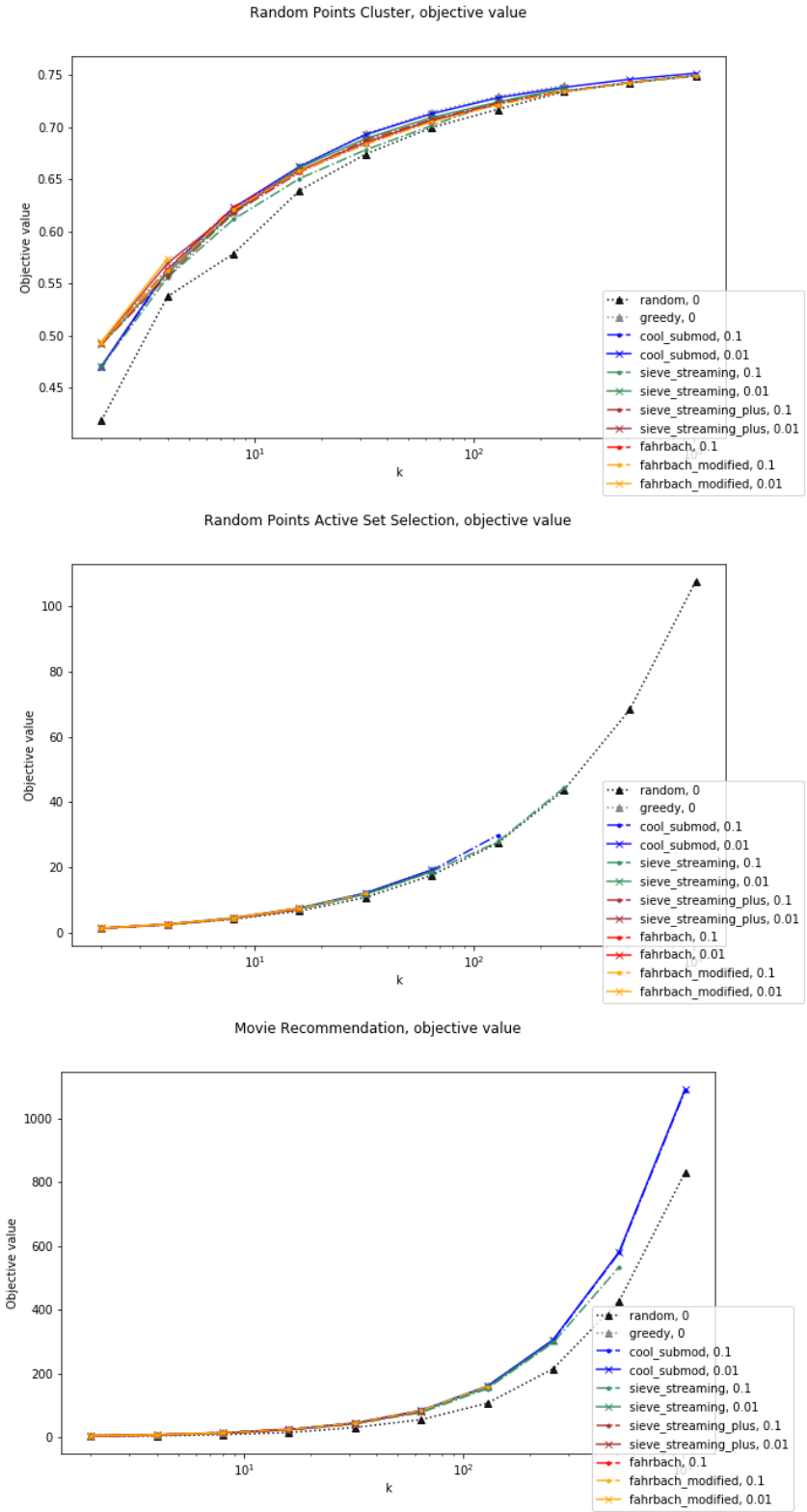


Figure 3.1: The objective values from different submodular maximisation problems computed using different algorithms for varying $k$.

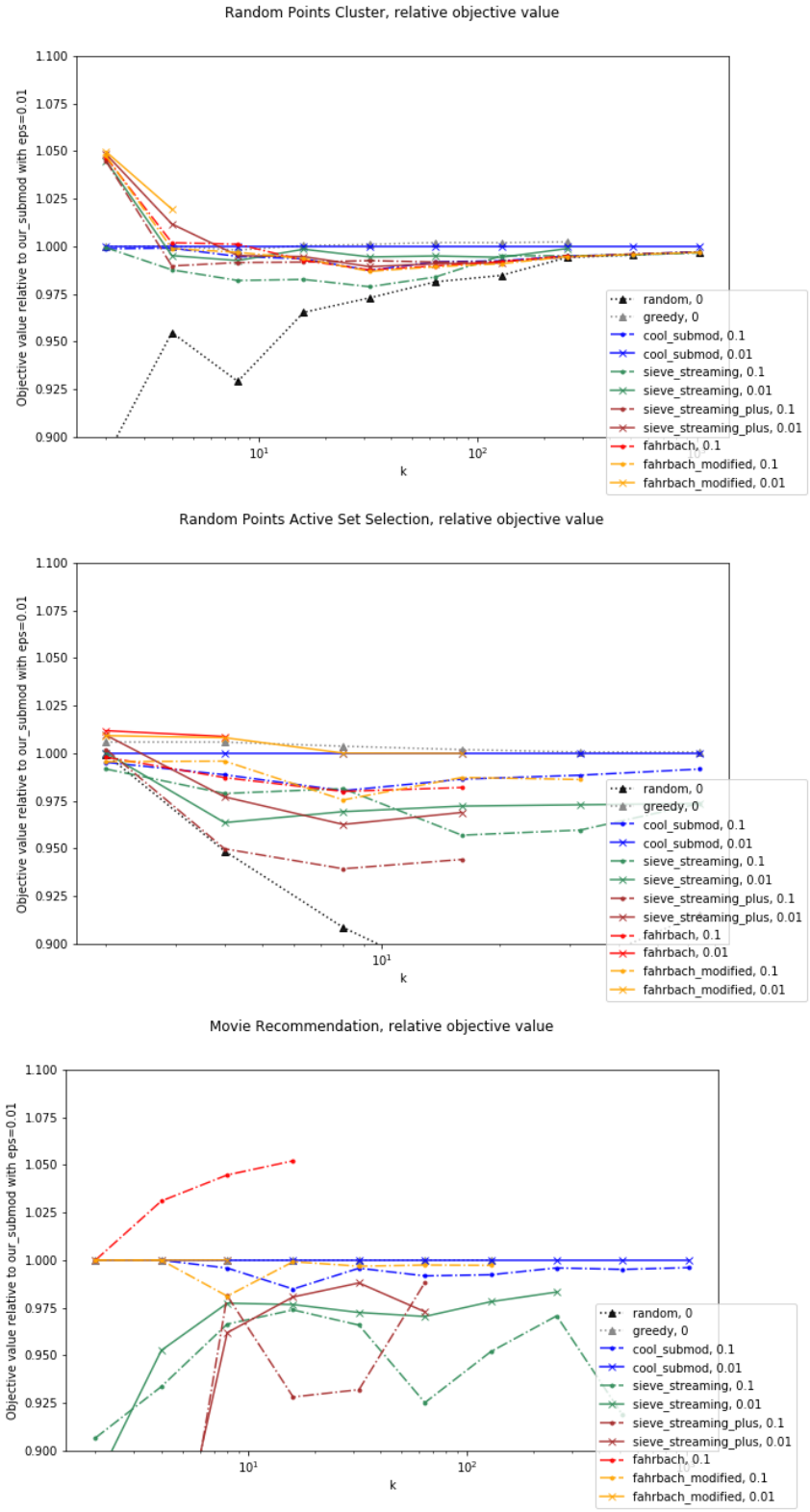


Figure 3.2: The relative objective values from different submodular maximisation problems computed using different algorithms for varying $k$. The values are all scaled to be relative to the objective values obtained from COOL-SUBMOD ran for $\varepsilon = 0.01$.

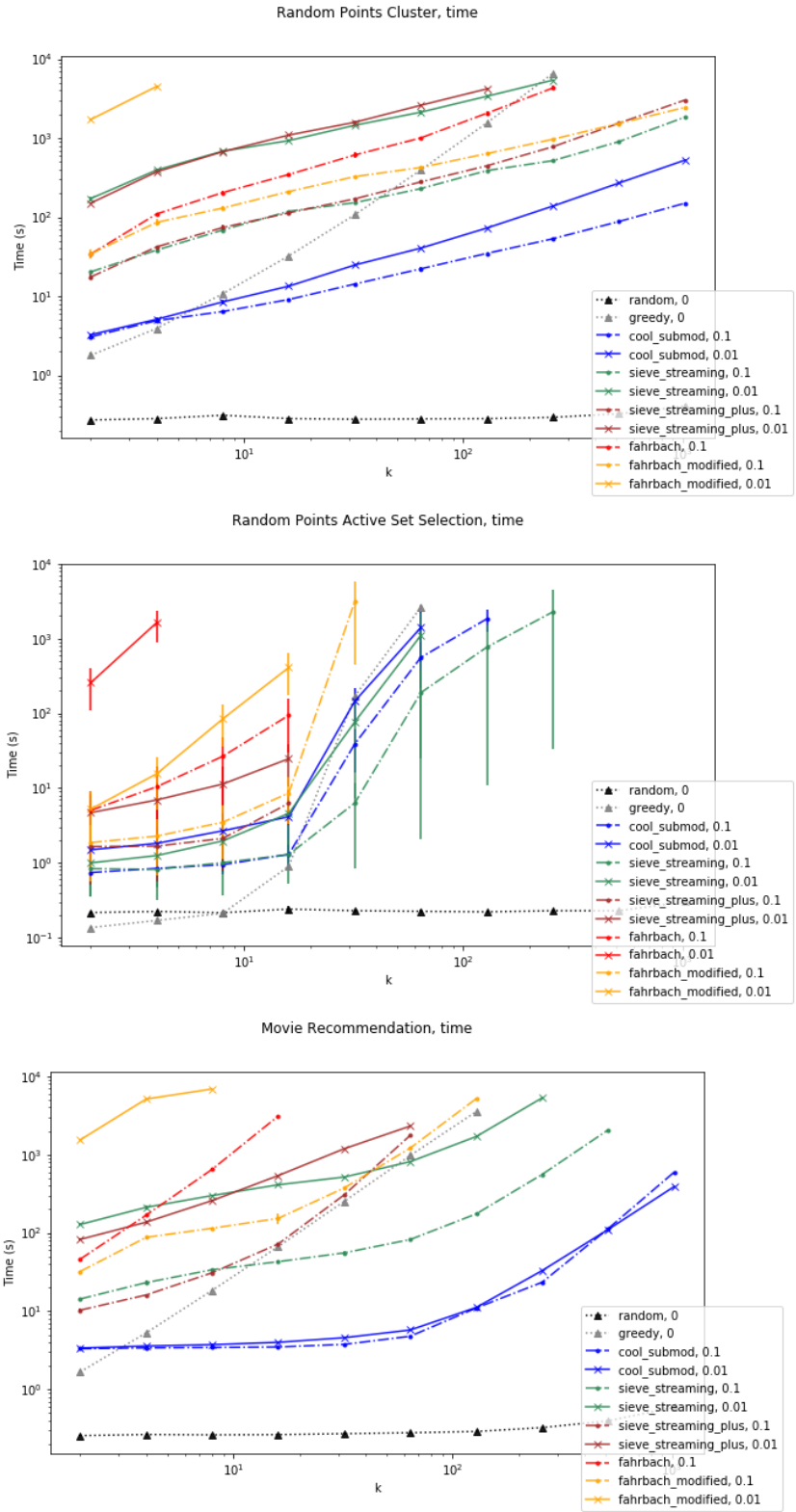


Figure 3.3: The time required to compute different submodular maximisation problems using different algorithms for varying $k$. All tests were run on four threads.

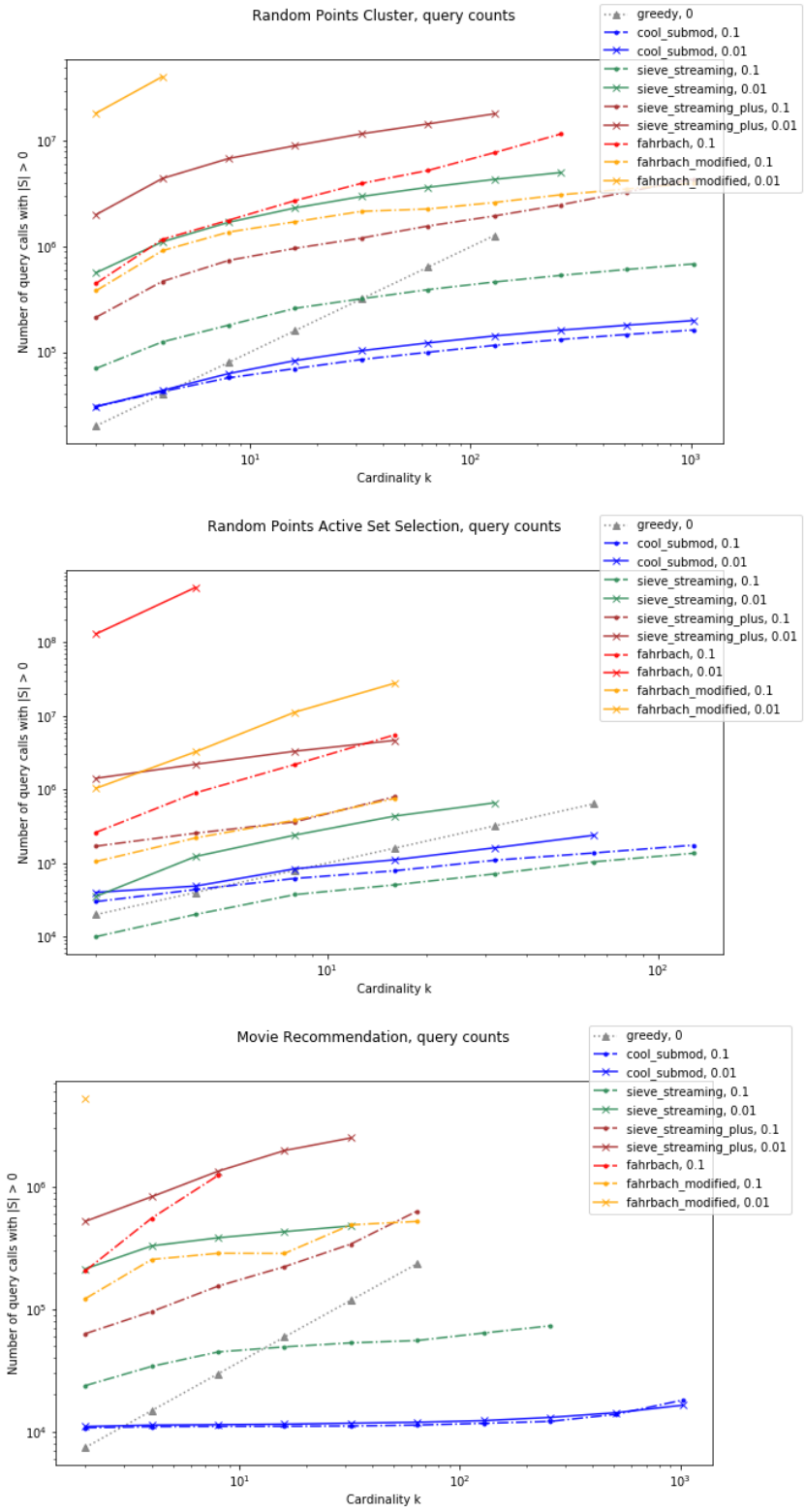


Figure 3.4: The number of query calls for solving different submodular maximisation problems using different algorithms for varying $k$.

round, which would allow us to better the performance of our algorithm.

We also believe the algorithm could be optimised further, specifically in the THRESHOLD-SAMPLING method. At the moment, this subroutine is the main bottleneck of the algorithm, and so far is a subroutine we have taken from a previous work. If we are able to further optimise this subroutine, it could provide an even better algorithm than what we have presented so far.

# CHAPTER 4

# CONCLUSION

## 4.1 Summary

We summarise the details of the work in this thesis as follows.

### 4.1.1 Sentence Diversification for Machine Translation

In Chapter 2, we explored methods of sentence diversification for selecting a training set for machine translation tasks. We have introduced different distance metrics and algorithms we are able to use to perform our diversification tasks.

We constructed a framework to perform sentence sampling and tested the performance of the system on providing a training set for machine translation tasks. Here, we tried to perform diversification on a large corpus, and then trained a machine translation model using the selected sentences. We were able to achieve better-than-random performances in some diversification setup using token-based distances.

### 4.1.2 Submodular Maximisation For Large Datasets Under Cardinality Constraint

In Chapter 3, we explored the problem of submodular maximisation under cardinality constraints. We examined existing parallel algorithms, which all tend to require large runtime in practice and large number of parallel function calls, which is not practical when implemented.

To counter this, we developed our on algorithm which is aimed to be practical. We found that our algorithm performed far better than theoretical parallel algorithms due to its lower runtime and number of function queries, while also providing solutions which are guaranteed to be as good as the existing algorithms. To confirm this, we ran experiments on our algorithm and other parallel algorithms using both artificial and real dataset.

## 4.2 Future Extensions

We find that while we have performed tests on works from both sections, more extensive testing would be needed to make the results more credible. For our sampling system, we would like to test on more corpuses, while in the submodular maximisation algorithm, we would like to perform tests on more submodular functions.

We also would like to better understand the results we have obtained. In the sampling system, we find that there are still more hypotheses that can be examined with why some distance function works well than the others. In our submodular maximisation algorithm, there remains details in the proofs which have not been completely ironed out. Being able to understand the results and the theoretical guarantees further will allow us to have more methods to improve upon the works we already have.

We also believe improvements upon the submodular maximisation algorithm can be useful for our sentence selection process, as they may provide good algorithms for diversification according to our desired objectives. If we are able to construct a meaningful diversification setup for our sentence selection problem, and are able to compute them using our submodular maximisation algorithm, we believe it can be an exciting system which can be useful for machine translation problems.

# APPENDIX

# APPENDIX A

# OMITTED DETAILS FROM CHAPTER 3

## A.1 A Faster REDUCED-MEAN Algorithm

Our version of REDUCED-MEAN will only sample from the Bernoulli distribution $\mathcal{D}$ in small increments, and will also terminate early in the case that we are confident that our distribution have the required expected values.

**Algorithm 13** An improved REDUCED-MEAN algorithm

**function** ADJUSTED-REDUCE-MEAN($\mathcal{D}$, $\varepsilon$, $\delta$)
    $\alpha \leftarrow \ln(2/\delta)/2$
    $k \leftarrow 1$
    $t \leftarrow 0$ ▷ Number of samples so far
    $X \leftarrow 0$ ▷ Number of times we sample a 1
    **while** $k \leq 16/\varepsilon^2$ **do**
        Sample $X_{t+1}, X_{t+2}, \ldots, X_{t+k\alpha} \sim \mathcal{D}$ in parallel
        $t \leftarrow t + k\alpha$
        $X \leftarrow \sum_{i=1}^{t} X_i$
        $\beta \leftarrow 1 + 1/\varepsilon\sqrt{k}$ ▷ For a tighter lower bound
        $\gamma \leftarrow 2 - 1/\varepsilon\sqrt{k}$ ▷ For a tighter upper bound
        **if** $X/t \leq 1 - \beta\varepsilon$ **then**
            **return** `True`
        **else if** $X/t \geq 1 - \gamma\varepsilon$ **then**
            **return** `False`
        $k \leftarrow 2k$
    **if** $X/t \leq 1 - 1.5\varepsilon$ **then**
        **return** `True`
    **else**
        **return** `False`

We will first prove an invariance about our algorithm.

**Claim A.1.1** (Correctness of ADJUSTED-REDUCED-MEAN)**.** *Let $\mu$ be the mean of the Bernoulli distribution $\mathcal{D}$. With probability of at least $1 - \delta$, for any $k \geq 1$, the following holds for an iteration of the while loop.*

1. *If* ADJUSTED-REDUCED-MEAN *returns* `True`, then $\mu \leq 1 - \varepsilon$.
2. If ADJUSTED-REDUCED-MEAN returns `False`, then $\mu \geq 1 - 2\varepsilon$.
3. If ADJUSTED-REDUCED-MEAN does not break during the while loop, then no conclusions are drawn about $\mu$.

*Proof.* We have to show that ADJUSTED-REDUCED-MEAN will wrongly return an answer with the probability of $\delta$. There are two possible scenarios where this occurs - when ADJUSTED-REDUCED-MEAN returns `True` but $\mu > 1 - \varepsilon$, or when ADJUSTED-REDUCED-MEAN returns `False`, but $\mu < 1 - 2\varepsilon$.

Consider the first case, where ADJUSTED-REDUCED-MEAN returns `True` but $\mu > 1 - \varepsilon$. We see that in order for the algorithm to return `True`, we require that $X \leq t(1 - \beta\varepsilon)$. Assuming that $\mu > 1 - \varepsilon$, we see that

$$\begin{aligned}
\Pr[X \leq t(1-\beta\varepsilon)] &= \Pr[X - \mu t \leq t(1 - \beta\varepsilon - \mu)] \\
&\leq \Pr[X - \mu t \leq t(1 - \beta\varepsilon - (1-\varepsilon))] && \text{since } \mu > 1-\varepsilon, \\
&= \Pr[X - \mu t \leq \varepsilon t(1-\beta)] \\
&\leq \exp\left\{\frac{-2(\varepsilon t(1-\beta))^2}{t}\right\} && \text{by Hoeffding's Inequality,} \\
&= \frac{\delta}{2} && \text{by our choice of } \beta.
\end{aligned}$$

A similar analysis can be done on the second case when ADJUSTED-REDUCED-MEAN returns `False` but $\mu < 1 - 2\varepsilon$. In order for the algorithm to return `False`, we require that $X \geq t(1 - \gamma\varepsilon)$. Assuming that $\mu < 1 - 2\varepsilon$, we see that

$$\begin{aligned}
\Pr[X \geq t(1-\gamma\varepsilon)] &= \Pr[X - \mu t \geq t(1 - \gamma\varepsilon - \mu)] \\
&\leq \Pr[X - \mu t \geq t(1 - \gamma\varepsilon - (1-2\varepsilon))] && \text{since } \mu < 1-2\varepsilon, \\
&= \Pr[X - \mu t \geq \varepsilon t(2-\gamma)] \\
&\leq \exp\left\{\frac{-2(\varepsilon t(2-\gamma))^2}{t}\right\} && \text{by Hoeffding's Inequality,} \\
&= \frac{\delta}{2} && \text{by our choice of } \gamma.
\end{aligned}$$

In our algorithm, the two cases occurs with a probability of at most $\delta/2$ each, therefore by union bound, our algorithm will fail with a probability of at most $\delta$. □

From the invariance above, we can see that if the algorithm terminates early, then with a probability of at least $1 - \delta$, the returned value correctly indicates whether $\mu \leq 1 - \varepsilon$ or $\mu \geq 1 - 2\varepsilon$, and otherwise will run more tests until it is confident that one of the condition holds.

Finally, we can give a theoretical bound for the efficiency of the algorithm.

**Claim A.1.2.** ADJUSTED-REDUCED-MEAN *requires* $O(\ln(1/\delta)/\varepsilon^2)$ *oracle queries and has overall adaptivity of* $O(\ln(1/\varepsilon^2))$.

We note that while the adaptivity of this algorithm is no longer constant and that the number of oracle queries does not decrease in the worst case, this algorithm will run faster in practice as the worst case will not often be reached.

# REFERENCES

[AN20] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor, 2020.

[BMKK14] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 671–680, New York, NY, USA, 2014. Association for Computing Machinery.

[BRS18] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. 04 2018.

[CQ18] Chandra Chekuri and Kent Quanrud. Parallelizing greedy for submodular set function maximization in matroids and beyond, 2018.

[CYK+18] Daniel Cer, Yinfei Yang, Sheng-Yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[DP12] Marina Drosou and Evaggelia Pitoura. Disc diversity: Result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment*, 6(1), 2012.

[EN18] Alina Ene and Huy L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time, 2018.

[EOAG18] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale, 2018.

[FMZ18a] Matthew Fahrbach, Vahab Mirrokni, and Morteza Zadimoghaddam. Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. *arXiv preprint arXiv:1808.06932*, 2018.

[FMZ18b] Matthew Fahrbach, Vahab Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity, 2018.

[HK15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[Hyy02] Heikki Hyyro. Explaining and extending the bit-parallel approximate string matching algorithm of myers. 10 2002.

[KG14] Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability*, 2014.

[KKD+18] Guillaume Klein, Yoon Kim, Yuntian Deng, Vincent Nguyen, Jean Senellart, and Alexander Rush. OpenNMT: Neural machine translation toolkit. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Papers)*, pages 177–184, Boston, MA, March 2018. Association for Machine Translation in the Americas.

[KMVV15] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Trans. Parallel Comput.*, 2(3), September 2015.

[KMZ+19] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity, 2019.

[KSKW15] Matt Kusner, Y. Sun, N.I. Kolkin, and Kilian Weinberger. From word embeddings to document distances. *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, pages 957–966, 01 2015.

[LSH02] Neil Lawrence, Matthias Seeger, and Ralf Herbrich. Fast sparse gaussian process methods: The informative vector machine. volume 15, pages 609–616, 01 2002.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[Min78] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In J. Stoer, editor, *Optimization Techniques*, pages 234–243, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[MSN11] Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. Incremental diversification for very large sets: A streaming-based approach. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 585–594, New York, NY, USA, 2011. Association for Computing Machinery.

[NFTM+18] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrović, Amir Zandieh, Aida Mousavifar, and Ola Svensson. Beyond $1/2$-approximation for submodular maximization on massive data streams, 2018.

[NWF78] George Nemhauser, Laurence Wolsey, and Marshall Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[PNI+18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. Bleu: a method for automatic evaluation of machine translation. 10 2002.

[Set10] Burr Settles. Active learning literature survey. Technical report, 2010.

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2015.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[Tie12] Jorg Tiedemann. Parallel data, tools and interfaces in opus. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.

[ZWQ+16] Kaiping Zheng, Hongzhi Wang, Zhixin Qi, Jianzhong Li, and Hong Gao. A survey of query result diversification. *Knowledge and Information Systems*, 09 2016.