Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

Efficiently identifying top k similar entities

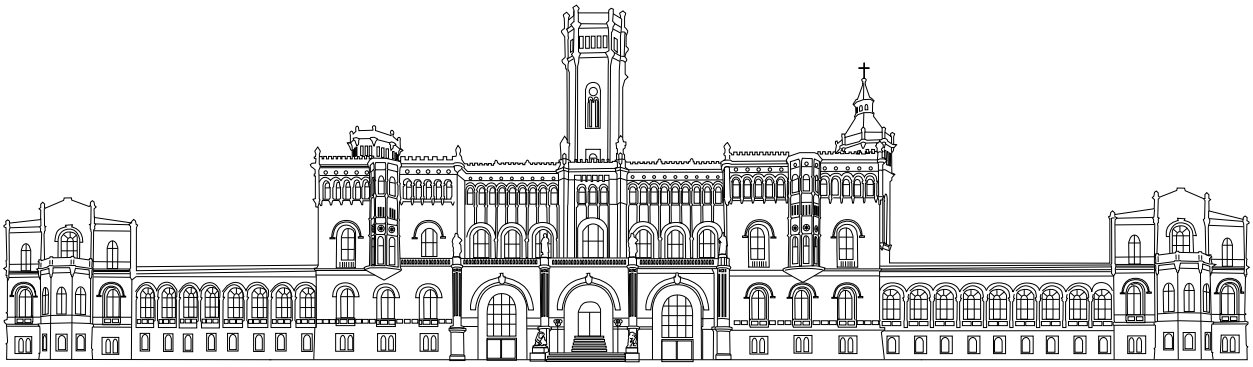
A thesis submitted in fulfillment of the requirements for the degree of
Master of Science in Internet Technologies and Information Systems (ITIS)

BY

Supreetha Hanasoge Sudheendra
Matriculation number: 10009763
E-mail: hanasoge.sudheedra@stud.uni-hannover.de

First evaluator: Prof. Dr. Sören Auer
Second evaluator: Prof. (Univ. Simon Bolivar) Dr. Maria Esther Vidal
Supervisor: Prof. (Univ. Simon Bolivar) Dr. Maria Esther Vidal

December 12, 2020



Declaration of Authorship

I, Supreetha Hanasoge Sudheendra, declare that this thesis titled, 'Efficiently identifying top k similar entities' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Supreetha Hanasoge Sudheendra

Signature: _____

Date: _____

"To action alone hast thou a right and never at all to its fruits; let not the fruits of action be thy motive; neither let there be in thee any attachment to inaction."

— Shrimad Bhagawad Gita, Chapter 2, verse 47

Acknowledgements

First and foremost, my deepest thanks to Prof. Dr. Sören Auer and Prof. Dr. Maria-Esther Vidal for giving me an opportunity to develop my thesis at TIB. I am extremely grateful for the unwavering support, guidance and mentorship that I received from Prof. Dr. Maria-Esther Vidal. Having her mentorship and to witness the accomplishments of the scientists at TIB has been an inspiration and one of the best outcomes of my masters' journey. Without the TIB family, the completion of my thesis would not have been possible.
I would also like to thank the scientists at TIB, Samaneh Jozashoori and Kemele M. Endris for the insightful discussions and for lending me a patient ear at various stages of development of my thesis.
I would like to thank my mother for supporting me in my decision to pursue my career aspirations no matter what and for providing me with the privilege that it takes to be here. This has been our dream, not just mine and I dedicate this to you.
Special thanks to my husband , Avinash Ramarao , the wind beneath my wings, for his invaluable support , for always pushing me to never give up and to have always had the faith in my abilities and progress.

Supreetha Hanasoge Sudheendra

Abstract

With the rapid growth in genomic studies, more and more successful researches are being produced that integrate tools and technologies from interdisciplinary sciences. Computational biology or bioinformatics is one such field that successfully applies computational tools to capture and transcribe biological data. Specifically in genomic studies, detection and analysis of co-occurring mutations is an leading area of study. Concurrently, in the recent years, computer science and information technology have seen an increased interest in the area association analysis and co-occurrence computation. The traditional method of finding top similar entities involves examining every possible pair of entities, which leads to a prohibitive quadratic time complexity. Most of the existing approaches also require a similarity measure and threshold beforehand to retrieve the top similar entities. These parameters are not always easy to tune. Heuristically, an adaptive method can have wider applications for identifying the *top most* similar pair of mutations (or entities in general). In this thesis, we have presented an algorithm to efficiently identify top k similar pair of mutations using co-occurrence as the similarity measure. Our approach used an upperbound condition to iteratively prune the search space and tackled the quadratic complexity. The empirical evaluations show that the proposed approach shows the computational efficiency in terms of execution time and accuracy of our approach particularly in large size datasets. In addition, we also evaluate the impact of various parameters like input size, k on the execution time in top k approaches. This study concludes that systematic pruning of the search space using an adaptive threshold condition optimizes the process of identifying top similar pair of entities.

Keywords: Bioinformatics, genomic studies, similarity, co-occurrence computation, time complexity, algorithm

Contents

List of Figures

List of Tables

Acronyms

FA Fagin's Algorithm

OEM Optimized exhaustive method

SOTA State Of The Art

TA Threshold Algorithm

TOP-MATA Top-k cOsine similarity Pairs using MAx-first TrAversal method

Chapter 1

Introduction

Co-occurrence of genetic mutations is an important biomarker in identifying characteristics of certain tumour conditions during the diagnostic and prognostic analysis of patients. In the field of genomics, specifically in genomic studies, presence and/or absence of certain mutations is an integral part of clinical and academic studies. Cancer, for example, is regarded as a primary result of genetic mutations that exert their effects on proto-oncogenes. As explained in [1], several environmental factors can lead to these proto-oncogenes to change to an oncogene which can cause cancer either in the form of a tumour or metastases. Every cancer sub-type is caused by a few mutations that co-occur in combinations. These combinations are reported to range from two-eight depending on the type of cancer [2]. Thus, evidence shows that combinations of mutations can be the cause of specific types of cancers, and therefore, identifying these combinations as important bio-markers can help in improved diagnosis, early reports of predispositions, and enhanced targeted drug treatments.

As reported in [1], the human genome contains approximately anywhere from 50,000 to 90,000 genes apart from those obtained from individual unknown sources. Several thousand mutations can be observed in relatively small number of tissue samples. This makes the computational aspect of identifying combinations of mutations highly complex. This research principally undertakes the problem of identifying top-k pair-

wise co-occurrences of different gene mutations in an efficient way. The findings could help in identifying specific biomarkers.The methodology proposed in this thesis focuses on developing a solution that aims to tackle the problem from the perspective of handling the high complexity of the search space that is typical of such computational challenges. Our method is compared with State-Of-The-Art (SOTA) methods of computing pairwise intersections as a baseline for empirical evaluations. Experimental outcomes suggest that the proposed method is observed to perform at higher levels of efficiency in time and accuracy.

1.1 Motivating example

The motivation for this work of research comes from more than two of the below perspectives: the first being numerous areas in which the problem of top-k retrieval is relevant. The wide-spread applications include the fields of document similarity detection , association rule mining in market basket analysis, association analysis for the study of drug-drug interactions, shelf management systems, alternate query formulations [18], climate studies [19] and the field of bioinformatics which is where our motivation for this study stems from and, many more fields have seen an increased interest in identifying top most similar item pairs. The second perspective comes from the scalability aspect, as to how can the problem be tackled computationally. We will see more on scalability in the upcoming paragraphs.The motivation for this study can be understood better with the help of the following example. Consider the following list of mutations as input 'list1' :

Mutations	Patient ID(s)	PatientCount
A	1,2,3,4,5	5
B	1,2,4,5	4
C	6,8,9	3
D	1,2,6,8,9	5

Table 1.1: Contents of 'list1'

The length of list1, i.e., the number of mutations in list1, is 4. The list contains information about four mutations that we call as A, B, C, D. Each mutation is

associated with different patients who are identified with single digit id(s) in the field 'Patient ID(s)'. Figure 1.1 illustrates this computational procedure.

Note : Within the context of this research, we shall refer to the data items as being 'mutations'. But, for the benefit of generalizability, it is important to note here that 'mutations' are a particular type of data entity. Therefore, it is interchangeably used with 'entities' in this work.

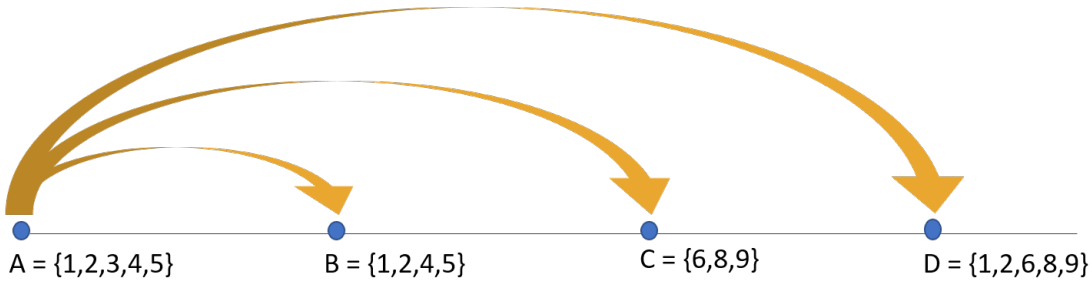


Figure 1.1: **Motivating example:** Representation of pairwise comparisons made using a brute force process of retrieving topk pairwise similar entities. As shown in this figure, every item starting with A is compared with every other item B,C and D for similarity computation.

In order to calculate the top 3 most similar pairs of sets, every mutation in list1 needs to be compared with every other mutation in list1, i.e., $(n-1)$ other items in total which is briefly represented in the above figure. Mutation A is shown as being compared with mutations B, C and D. Similarly, mutation B, C and D will be compared with the remaining 3 mutations. This way, for an input list of length 4 mutations, 12 comparisons are made before retrieving the top-k pairs that are most similar, i.e., $n(n-1)$. Out of these, since similarity as a measure, is symmetric, an effective number of comparisons amount to $n(n-1)/2$. Therefore, in the traditional approach referred to as the "brute-force" technique, the complexity of this problem is prohibitively in $O(n^2)$. This comes as a massive disadvantage to scalability of this solution to bigger datasets. Figure 1.2 presents an outline of the traditional approach that is typically unscalable for larger datasets.

As the number of mutations increases in the dataset, the amount of time required to evaluate the pairwise similarities or comparisons increases quadratically leading to sub-optimally performing solutions. The problem of efficiently retrieving top-k similar entities, or mutations in our context, depends on the number of mutations

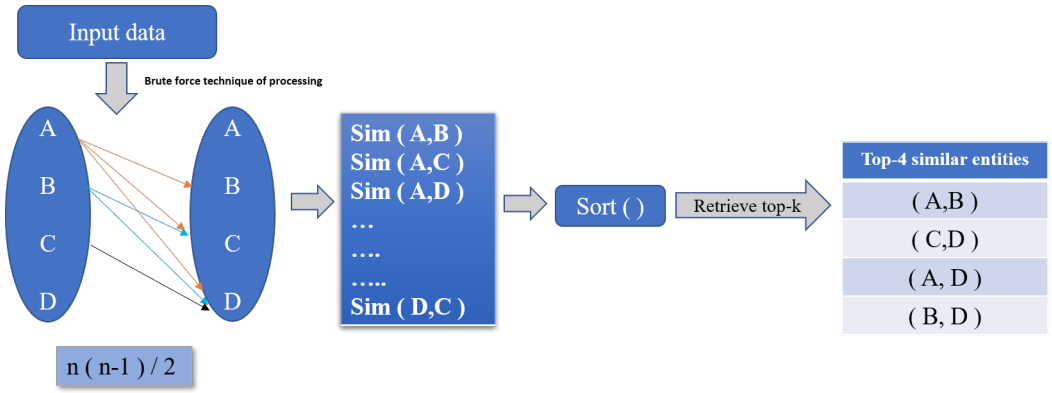


Figure 1.2: Example pipeline followed to extract top similar entities

present in the input data i.e the value of n, the value of k and depending on the similarity metric that is chosen, the efficiency of the solution also depends on the overall distribution of the number of patients that each mutation is associated with. As the value of n increases, we know that the number of comparisons increase in $O(n^2)$, which means that in order to decrease the time required to retrieve the top-k, we need to reduce the number of comparisons that need to be made or add more computational resources to iterate through the solution's search space. Within the boundaries of this work, our approach tackles this disadvantage of scalability using the principle of reducing the number of comparisons in order to optimize the process of retrieving the top-k similar entities.

1.2 Problem and contributions

The authors in [27] define association analysis as the task of identifying strongly related subset of items, given a set of objects and their observed co-occurrence information.This work presents an approach for minimizing the number of comparisons required to retrieve top-k similar entities and thereby improve the performance in terms of run time and also maintaining the required level of accuracy. In order to optimize this process, Zhang and Feugenbaum of [26] report that it is uneconomical to

compute the coefficients of all the pairs in the process of identifying the highly correlated pairs. This is especially applicable to use-cases where the number of correlated pairs is smaller than the total number of pairs.

Our approach employs a pruning technique with which the number of pairs to compare are reduced iteratively. One of the straightforward ways to solve this problem is by adding more computing resources that and employing parallel programming techniques. We can see that the work in [10] utilizes the map-reduce framework to solve this problem by breaking it down to efficiently search the solution space. This approach can improve efficiency by alleviating run time, however, it does not contribute to handling the complexity of the problem that is polynomial. In this work, we propose a solution that aims to decrease the time required to efficiently retrieve topk similar entities by handling the complexity of the problem through an adaptive upperbound-based pruning strategy. Our research objective is to design an approach to retrieve topk similar entities following the aforementioned strategy with the aim to improve performance in terms of time taken and accuracy. We then compare the performance of our approach to two other approaches that includes a baseline algorithm used to solve the problem of topk extraction. We know from [21] that a naive approach compares every pair of entities in the search space which leads to a prohibitively $O(n^2)$ time complexity where n represents the number of input objects. Other than the dependence on n, this research also contributes a study of the impact of two more factors on the execution time of the considered algorithm:

- The value of k in top-k.
- The overall size of associations of each object in the input or in other words, "cardinality" of each set in the input.

From [7], it is known that the run time of an algorithm is determined by the total number of elementary operations that are executed. Therefore, an important factor that can be used to understand the runtime of different approaches is by observing the count of iterations. As a further step, we also look into the count of iterations or loop count to study of efficiency in the considered approaches.

The remainder of this thesis is structured as follows. In chapter 2, we discuss some of the related tools and technologies that are preliminaries to the work done here. The following chapter 3, presents a survey of the related work done in the area of topk algorithms and their application areas. In chapter 4, the proposed solution is presented

with a description of the steps followed in chapter 5, detailing the implementation of the proposed approach. The next section, chapter 6 shows the experimental evaluations done in order to measure the performance of the proposed approach and compare the same with the performance of the SOTA approaches in the form of a comparative study and analysis. In chapter 7, we present the concluding remarks. It also includes recommendations and insights for the purposes of future work.

1.3 Summary of the chapter

To summarise, this chapter provides the scope of the work done in this research. The chapter also provides the motivating example in order to better understand the existing problem that this work aims to tackle. As part of this goal, additional contributions made during this study are covered under contributions.

Chapter 2

Background

This chapter introduces some of the related concepts and preliminaries that will be used within the confines of this work. The following section aims to provide a clear understanding of the concept of 'similarity', data access methodologies, and various technological fields where these concepts are leveraged. Important design dimensions to be considered when using top-k processing techniques include implementation levels, scoring functions, and data access methods. In the following sections, we will look at scoring functions and data access methods and their impact on top-k processing results.

2.1 Analysis of Algorithms

An algorithm can be informally described as a computational process that takes a set of values as input and produces a set of values as output. Therefore, algorithms are regarded as tools for solving computational problems.

As explained in [7], the correctness of an algorithm can be ascertained from its execution nature when for every instance of input, it produces the correct output. The analysis of an algorithm is done by measuring the amount of resources that the algorithm requires. Various resources like memory consumption, communication bandwidth, and computer hardware are accounted for primarily but it is computational time that is often used as a measure of efficiency. Therefore, due to these system specific parameters, intuitively, it is reasoned that multiple algorithms built to solve a particular problem can differ in efficiency. Consequently, it is necessary to have a measure that can be used to compare relative performances and for this, it is recommended to use asymptotic analysis.

We know that the time required for an algorithm is dependent on the input size. This size of input is regarded as the size of the problem, an integer that is used to describe the problem numerically. Like mentioned earlier, an algorithm can also be regarded as a tool or in other words, it is seen as a black box that takes input values and processes the same to provide as output the solution to the problem. Therefore, they can be formally expressed as a function of the problem size. The time required for an algorithm that is expressed as a function is referred to as the time complexity of the algorithm.

We understand from [7], whenever the size of input is large enough, such that, only the increase in run time is studied, it is referred to as the asymptotic analysis of algorithms. With this measure, it is possible to analyze the relative efficiency of algorithms that are used to solve a particular problem. With asymptotic analysis, the main aim is to determine how the execution of the algorithm changes with change in input size. The asymptotic notation denoted by O, is the upperbound of run time of the algorithm in the worst-case scenario. If the worst case running time of an algorithm increases linearly with problem size, the time complexity is said to be $O(n)$, where n represents the input size. If the upperbound run time of an algorithm is bounded by a constant time, the algorithm is said to have constant time, $O(1)$.

If the worst case running time of an algorithm is bounded by a polynomial expression in terms of the problem size, such an algorithm is said to have polynomial time, $O(n^k)$. An algorithm whose upperbound run time is quadratic is represented by $O(n^2)$. On account of the above known grounds, in our body of work, we analyse run time with increasing input to compare the efficiency in terms of time complexity, of our proposed approach with the chosen contemporary approaches to establish the hypothesis.

2.2 Similarity measures

Different applications of information retrieval and data mining algorithms use different measures to identify similar objects. Therefore, it is important to understand the concept of 'similarity' and the various measures that exist. In the following section, we take a look at the concept of 'similarity'.

Similarity/Distance measures have been widely used in the field information retrieval and data mining. A similarity/distance metric measures how close or farther apart are two objects. The data objects are defined or represented using different characteristics and the distance between the representations of the objects is an estimation of closeness or distance between them. This way, similarity or distance metrics are used as 'scoring functions' in identifying the "top-most" similar entities. The work in [13] examines some of the distance measures used in partial clustering of text documents. The authors in [17] outline conditions that need to be met for a metric to be considered as a distance metric. Suppose v1,v2 are to objects and d is the distance between them, then:

1. The first condition is that the value of the distance d cannot be negative, i.e., $d(v1, v2) \geq 0$.

2. The distance value is 0 if and only if the two documents are identical.

3. The distance measure is symmetric is, $d(v1, v2) = d(v2, v1)$.

4. The measure must satisfy the triangle inequality which is $d(v1, v3) \leq d(v1, v2) + d(v2, v3)$.

Document classification and clustering tasks often use binary representations for the documents called the term-document matrix. In a term-document matrix, every document in the corpus is represented by a row and every word (feature) in the corpus is represented by a column. Each value in a cell of the matrix, can contain either 0 or 1 which corresponds to absence or presence respectively. This way, every document is represented by an n-dimensional binary vector.

2.2.1 Simple Matching Coefficient (SMC)

A data object can be represented in terms of the absence and presence of a given set of features. Similarity measures between binary vectors can be evaluated by taking presence and absence of attributes into account. Consider two binary vector representations $v1$, $v2$ that are being compared. With binary representations there can be four different types of scenarios for a given feature dimension $y1$. They are,

S11 : where binary value of y1 is 1 in both vectors v1 and v2 - Mutual presence.

S00 : where binary value of y1 is 0 in both vectors v1 and v2 - Mutual absence.

S10 : where binary value of y1 is 1 in v1 and 0 in v2.

S01 : where binary value of y1 is 0 in v1 and 1 in v2.

The formula for SMC is as shown below:

$$SMC(v1, v2) = \frac{\text{number of matching attributes}}{\text{number of attributes}}$$

In other words,

$$SMC = \frac{\text{S00+S11}}{\text{S11+S00+S10+S01}}$$

Therefore, the SMC measure considers only two scenarios from the above enumeration, S11 and S00. This property makes SMC a suitable similarity measure when mutual absence of a feature in both vectors represented by S00, is as semantically meaningful as mutual presence , represented by S11. Chapters in future will show why the above concept is relevant in this work.

2.2.2 Hamming distance

From [21], we know that the concept of similarity is closely related to distance between the representation of two objects. By logic, the more similar given pair of

objects are, less is the distance measure between them. Hamming distance is therefore defined as the size of their symmetric difference. Given two objects A and B, the hamming distance is:

$$H(A, B) = |(A - B) \cup (B - A)|$$

Levenshtein or edit distance is another measure of *distance* [21].

2.2.3 Jaccard coefficient

In the context of text document representation, every word present in the document is considered as a dimension. Therefore, every document can be represented as a multidimensional vector. For any two documents represented using vectors $v1$, $v2$, the Jaccard coefficient can be defined using the following equation[21].

$$J(v1, v2) = \frac{|v1 \cap v2|}{|v1 \cup v2|}$$

The value of Jaccard coefficient lies between [0,1] where 0 represents dissimilarity among the documents which means they share no common terms and 1 represents the highest value of similarity among the documents. As shown above, the Jaccard coefficient is the ratio of the number of words/tokens shared by the two documents in consideration and the total number of unique tokens present in either documents. Therefore, the Jaccard coefficient is sensitive to the length of documents. Longer documents have higher probability of common words but in essence could actually be more semantically dissimilar. Such a use-case is not appropriately captured by the Jaccard measure. The difference between SMC and Jaccard lies in the fact that Jaccard coefficient only considers intersection in the numerator, i.e., only mutually present dimensions. This makes Jaccard a more general purpose measure in vectors that are not necessarily binary.

2.2.4 Cosine similarity

Cosine similarity [27] is a measure of cosine of the angle between vectors $v1$, $v2$ that represent two documents. The formula to calculate the cosine similarity between vectors $v1$ and $v2$ is as shown below:

$$\cos(v1, v2) = \frac{(v1 \cdot v2)}{(||v1|| \times ||v2||)}$$

For non-negative vectors, the value of cosine similarity is bounded between [0,1]. A cosine similarity value of 0 means that the vectors are orthogonal to one another. This implies dissimilarity between them. Whereas, a cosine value of 1 implies that the vectors are parallel or are closely similar. One of the advantages of using cosine similarity is that it is independent of the size of the documents being compared. This measure is used to capture the orientation of the vectors in space rather than being a comparison of the magnitudes of the vector representations. Cosine similarity is also effective in ignoring zero-matches which is especially important in the context of information retrieval where numeric vectors are typically sparse in nature.

2.2.5 Euclidean distance

As reported in [13], the Euclidean distance measure is one of most used measures of similarity evaluation in clustering problems, K-means being one of the example algorithms. Euclidean distance between two documents x and y is given by the following equation, where n is the number of unique dimensions or features present in the corpus and x_i represents the weight of term i in document x and y_i represents the weight of term i in document y.:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2.2.6 Overlap measure

According to [21], overlap measure without normalization is defined as given two sets A and B, the overlap between the sets is given by the following:

$$|A \cap B|$$

This is nothing but the number of set elements that are common in both sets A and B. Bound of overlap measure Ov is:

$$\Big\{ 0 \leq Ov \leq min\,(\,|A|, |B|\,) \Big\}$$

In this thesis, the measure used to identify 'similar' entities is by counting the frequency of pairwise co-occurrences of mutations in patients. Therefore, we use the Overlap measure that is not normalized.

2.3 Data Access

The authors in Ilyas et al. [14] survey various top-k processing techniques can be differentiated based on the type of access methods that they assume to be available in the data sources they use. The Figure 2.1 provides an overlay of this differentiation. The data access methods can be of the following different types:

- No random;
- Both sorted and random; and
- sorted+ controlled random probes.

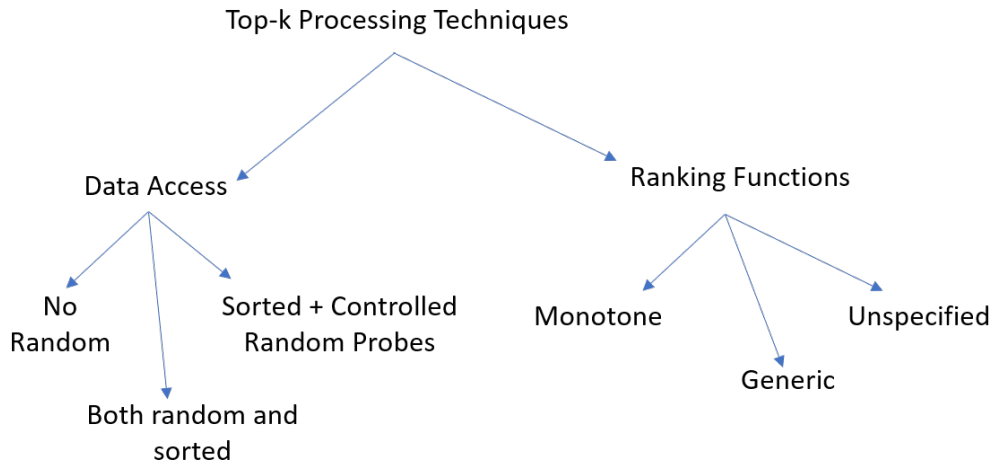


Figure 2.1: Part of the segregation of top-k data processing techniques as depicted in [14]

The performance and efficiency of top-k processing techniques are largely dependent on the data access methods. By principle, we know that random access is more expensive than sorted access. In the first type of data access method, the top-k processing technique assumes that the underlying data source supports both sorted as well as random access methods. Fagin et al. [12] propose the Fagin's Algorithm (FA). In FA, items are ranked using multiple ranking functions each creating multiple lists. The items in the lists are accessed parallely, with each access being sequential in nature. The operations stop when k different objects are seen in all the lists. In the following step, the total score is calculated for all the 'seen' or processed items from which the top k are selected through random access. The example in Figure 2.2 shows the process of FA.

In TA, similar to FA, a set of items are sorted based on multiple scoring functions. An upperbound value based on a scoring function is used to select the items. For every new object seen/processed in either of the lists, a new threshold is calculated based on the score of the seen objects. In the next step, for all the objects processed until now, the total score is calculated, which is performed through random access. Only those objects whose total score is greater than the threshold, get selected as output for top-k. Once k number of objects are selected, the algorithm operation terminates. This way, upperbound based selection and iterative pruning is used to efficiently identify top-k items in query processing applications. As explained in [14], in TA, the differences in cost for random access compared to sorted access has not

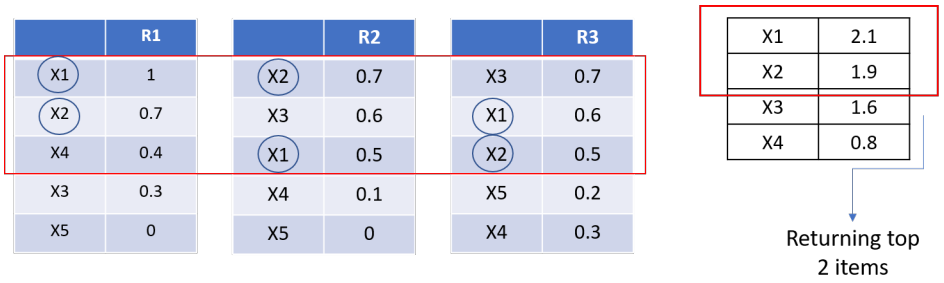


Figure 2.2: In the above example, for a k=2, the lists are ranked according to ranking functions R1,R2 and R3. Parallely, items are accessed sequentially in each list. Once k objects X1 and X2 are seen in all the lists, for all the objects 'seen' which are X1,X2,X3 and X4, the scores are computed by random access and the top 2 are retrieved.

been taken into consideration. Consequently, in the worst case, the TA approach generates a random access after every sorted access that produces a new object, and therefore, if there are m number of ways in which the objects were ranked leading to m lists, every sorted access with new object can lead to $(m-1)$ random accesses. Depending on the number of objects, this can lead to very expensive overall cost. The example in Figure 2.3 after referring [14], serves to understand TA.

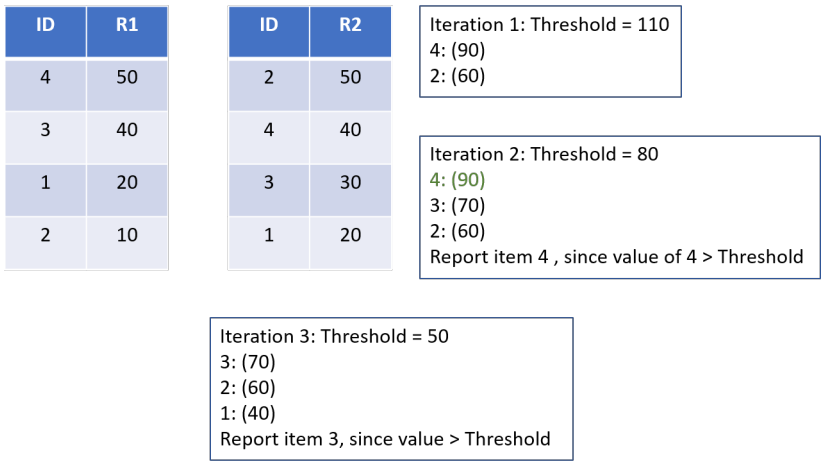


Figure 2.3: Working example of TA. If the value of k=2, the process stops after the iteration 3, and returns items 4 and 3 as top 2 query answer.

2.4 Evaluation measures

In the domain of information retrieval, the effectiveness of a classifier is assessed by measuring how well the system results match the user's query intent, which is quantified using relevance judgments. Evaluation measures like precision, recall, classification accuracy are used to provide a quantitative assessment of an information retrieval system relative to the query being answered. In the further passages, we will look at some of the widely used measures, and then discuss how they can be used in the context of retrieved top-k result verification. In the context of information retrieval, recall is the proportion of the relevant documents retrieved, and precision is the proportion of the retrieved documents that are relevant. Most ranking systems use a series of relevance judgments against the query, which is used cumulatively as input to quantitatively measure the ranking effectiveness. Recall and precision can also be measured at each document retrieved.

The Venn diagram in Figure 2.4 from [16] helps to better understand the concepts of precision and recall. It is to be noted that precision and recall are converse in nature. With a high recall, precision gets lower and vice-versa.

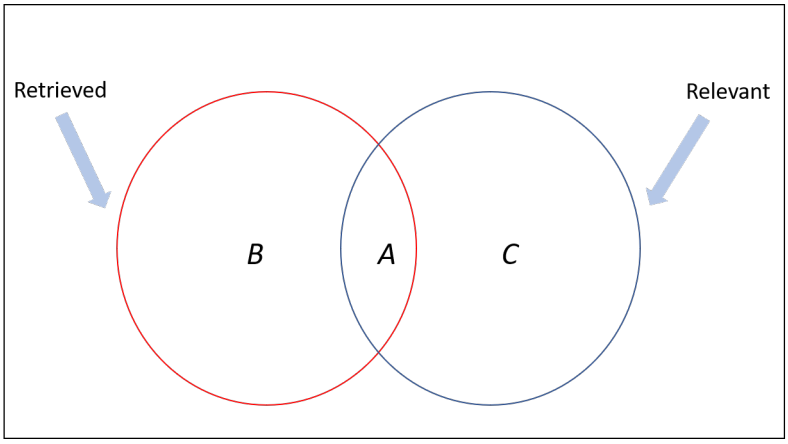


Figure 2.4: The Cranfield arrangement from [16]

From the fig Figure 2.4, precision is defined as $|A|/(|A|+|B|)$ and recall is defined as $|A|/(|A|+|C|)$. Within the context of IR, Precision is used to measure the efficiency of identifying true positives or in other words referred to as true positive rate. The

below figure Table 2.1 of a contingency table gives a brief overview distribution of different classes of results in a binary classification task. We shall use the concepts from this distribution in further sections for better understanding.

	Actual classes	
Predicted classes	Positive	Negative
Positive	True positive	False positive (type I error)
Negative	False negative (type II error)	True negative

Table 2.1: Contingency table

The recall measure is also referred to as "sensitivity". Recall is used to evaluate the efficiency of identifying all the records relevant to the query being answered. Applications that require high recall, are those for which identifying all relevant objects i.e. without any false negatives is of priority irrespective of how many irrelevant objects get retrieved. However, in the context of information retrieval, relevance is a measure that is calculated as a consequence of human judgement. Algorithms can be designed to either have high precision or high recall depending upon the requirement. For applications designed to have high recall, it is essential that the user is cognizant of all the relevant answers pertaining to the query, a priory. Pertaining to this, the authors in [6] opines that the presence of unexamined documents has always been part of performance evaluation measures. However, in case of high recall applications, if a user has no awareness of a particular document, then even if this document is relevant to the user's search, it makes no difference in that search. This makes precision a more straightforward evaluation measure. Also, for systems with high scalability requirements, this is a disadvantage. In our context, the aim of the proposed solution is to be able to retrieve all the top k similar entities from the "unexamined" collection of quadratically growing search space. Therefore, the disadvantages associated with respect to evaluation can be seen as two folded:

- It is imperative that the results be 100% accurate as the concept of "relevance" is not pertinent in our context.
- There is no scope for false positives. The k items retrieved by the algorithm, need to all be true positive.

In order to use the metric of recall in our context, if we interpret the task of retrieving top-k as a binary classification task, the pair of top similar entities belong in the class

of positives and the remaining pairs of entities can be regarded as belonging to the class of negative. With the aforementioned disadvantages, even though recall is used to evaluate the efficiency and correctness of the proposed solution, the goal is for the proposed approach to have 100% recall. This is because *approximations* in biomedical computational services are less desirable. Therefore, to evaluate performance, we opt to measure the sensitivity of the algorithms to establish completeness of the proposed approach as compared to the baseline.

2.5 Summary of the chapter

In summary, this chapter on background, provided an introduction to the technologies and concepts that help in better understanding the motivation of this work, the problem that we aim to tackle and also the design decisions that have gone into developing the proposed approach. We also discussed some fundamental evaluation measures to understand the concept of similarity, how it can be adapted in our work and the reasoning for our choice of measure.

Chapter 3

Related Works

The applications of top-k processing techniques are ubiquitous in nature. The growing volume of structured and unstructured data being generated warrants the need for higher levels of optimization as well as techniques that can be adapted to heterogeneous data. Optimization calls for computational simplification. In this direction, promising work has been done and we will present how the problem of topk can be handled and also highlight the domain-specific requirements.

3.1 Scalability oriented approach

One of the most extensive applications of top-k algorithms is in the field of association rule mining. Association rule mining plays an important role in the field of market basket analysis. Market basket analysis involves computational methodologies to quantify customer preferences and detect patterns in sales that are further used for promotion, shelf management and marketing.

The approach discussed in [27] aims to overcome disadvantages of the traditional support and confidence framework in association analysis by using the cosine similarity measure to identify top-k co-occurring items as part of association analysis. Most of the approaches require a minimum threshold value of the distance measure measure used and this can be a challenge in real world scenarios dealing with high volume and/or high dimensional data. In order to circumvent this challenge the authors in [27] proposed a novel method based on Max-First traversal, an algorithm named TOP-MATA to retrieve top-k similar item pairs computed using cosine similarity measure. This approach performs upperbound pruning of the search space using the monotone property of cosine similarity measure. The advantage of this method as shown by the authors is that it reduces the number of false-positive computations by pruning the item pairs without having to calculate the actual similarity value for all possible pairs.

One of the properties of this approach is that the upper bound is based on the cosine similarity that is expressed in terms of support measure used in frequent pattern mining. The approach uses current minimum as threshold to produce a filtering effect and properties of sorted item matrix as pruning effect. In the max-first traversal strategy, the iterations are limited to the upper triangle of the sorted item matrix and rows are selected based on the presence of the item pair with maximum cosine upper bound. Upon selection of the row, all pairs in the row having the same value of cosine upper bound are investigated.

A positive aspect of this approach is that it limits the search space to $\frac{n(n-1)}{2}$ which is half in comparison to the naive approach of investigating every item with every item in the input. This leads to improved time efficiency. With the help of the upper bound, it also ensures that the pool of false-positive pairs that are investigated reduces, leading to performance enhancement by further alleviating the computational costs. In both TOP-MATA and proposed approaches, sorted inputs are used which ensures faster pruning [21].The motivation behind both approaches also coincide in that the solutions are driven around the need for efficiently retrieving top similar items without the presence of an overall minimum threshold value. Both approaches use the current upperbound to iteratively reduce the search space complexity. The main difference of this approach from the proposed approach is the usage of a property for the pruning effect. The monotone property of the cosine similarity measure is used to filter out candidate item pairs and only those pairs are investigated further at every iteration.

In the proposed approach, the current minimum value of co-occurrence in the current

top-k is used directly to prune the search space with the aim to reduce the complexity further in each iteration. In order to adapt the approach presented in [27] to the research problem of our undertaking, "minimum" of property is used. The sorted item-matrix is constructed by inserting the minimum cardinality of the two items as shown below:

$$card(item1, item2) = min(\ |\ \text{cardinality of item1}\ |, |\ \text{cardinality of item 2}\ |\ )$$

The work in this study and our proposed approach can be understood better with the help of the following high level comparison reported in Table 3.1.

Property	TOP-MATA	Proposed approach
Data input format	Sorted item matrix where each value is given by the upper-bound condition.	Sorted table of columns mutations, associated list of patients and the column for cardinality of this list of patients
Upperbound condition	In terms of support values of the pair of entities	In terms of the minimum() of the cardinalities of the pair of entities
Traversal	Uses max-first diagonal traversal starting from the row with the highest value in the item matrix	Using combinations of mutations retrieved after sorting the mutations in descending order of their cardinalities
Optimization strategy	Selective processing of pairs in every row of the sorted item-matrix	Iterative pruning of the input data using the updated value of threshold
Similarity metric	Cosine similarity	Overlap coefficient

Table 3.1: Outline of the major differences between TOP-MATA and Proposed approach

3.2 Computationally Economic Approach

Another one of the applications of efficient top-k retrieval has been in the field of document similarity assessment for clustering and information retrieval. The work in [17] defines the problem of document similarity computation as the process of computing the topk pair of similar documents using a specific term weighing model given a collection c where each document is represented by d.

This study proposes the use of shingles having a user-specified length to extract candidate pairs, where a 'k-shingle' for any document is defined as any substring of length k found within the document. Shingles are used to represent documents as a set of consecutive substrings that appear in the document. If the value of k is 4, then the document is divided into sets of 4 unique substrings that appear consecutively in the document. A given computational disadvantage that the models have to overcome is their quadratic runtimes. If a given dataset contains n documents, then, computing pairwise document similarities implies comparing every document with the remaining $(n-1)$ documents. The runtime with this approach increases quadratically with n, i.e., $O(n^2)$. The proposed model in this approach tackles the problem of high computational costs by employing heuristic methods to reduce the number of candidate comparisons. The shingles that each document is associated with, play an important role in the computation time. This is because each document can be associated with a large number of shingles leading to quadratic time complexity. The large number of shingles could also imply high storage complexity as it would require external memory. In this work, the authors have designed an algorithm to create relation triples of the form (U,V,W) where document U contains shingle V and shingle V is also present in document W and are to be used as candidates in the similarity evaluation process. The process of generating these relation triples involves pruning the documents to be compared by reducing the number of shingles that are remaining thereby reducing the overall runtime required to generate pairwise document similarities. The proposed solution performs in logarithmic time complexity $O(nlogm)$ where m is the number of documents and on an average reduces the number of comparisons by 71% compared to the Allpairs search algorithm by [4] thereby increasing time efficiency. This work is similar to our approach in that it aims to reduce the runtime costs by aiming to reduce the problem complexity instead of adding more computing resources to solve the problem of identifying top-k. Another similarity with respect to design is that both the solutions involve reducing

the number of candidate pairs by heuristically limiting the number of comparisons to be evaluated. Both the solutions also consider data pre-processing in order to save the space complexity of our approaches. A characteristic matrix is a data structure where each row represents a document and columns represent different tokens present in the corpus. A cell in such a matrix can have either 0 or 1 as a value where 1 implies presence and 0 implies absence. Such characteristic matrices are usually sparse owing to the fact that the number of tokens in a corpus are always exponentially greater than the number of documents in the corpus.

In both solutions, the number of comparisons to be made are further reduced by sorting the input in decreasing order of size. The solution design in both cases reduces the input data size based on frequency and uses a relation data structure to store information from only those cells from the characteristic matrix where the value is 1. However, the difference between our work and the proposed solution in [17] is that the problem domain in both cases are vastly different. The distance measure used for identifying top-k is also different as our aim is to identify actual co-occurrences as *similarity* in a pair of mutations in terms of patients is semantically meaningless. The choice of distance measure also leads to a different , it pruning criteria.

3.3 Filtering principle for pruning

As mentioned earlier, topk algorithms have diverse applications in the field of knowledge discovery. One such application in the field of text mining is for near duplicate detection. The rising growth of data on the internet is leading to more complex challenges like data integration from heterogeneous sources. One of the issues in this domain is redundancy and this is where near duplicate detection techniques become highly relevant. Near duplicate documents have a high degree of similarity quantitatively but have different bitwise. Therefore, even though they are semantically equivalent, they are different in terms of representation. Identifying near duplicates efficiently can lead to more focused web crawling application, increased diversity and accuracy of query results, spam detection, detecting plagiarism, community mining in social networking sites etc. This leads to the requirement of identifying near duplicate records efficiently. The work of research in [21] aims to identify pairs of records

that have a similarity value that is at least equal to a given similarity threshold.

This study proposes an approach based on the principle of positional filtering of tokens in a record to efficiently identify near duplicates quantitatively using Jaccard similarity measure. This study proposes an algorithm that additionally includes suffix filtering principle to prune candidate records efficiently. The first stage is a candidate generation stage where potential records are extracted which are essentially the superset of the final set. In the next stage called the optimized verification algorithm, the actual similarities are computed and the near duplicates are identified using a given similarity threshold. In positional filtering, the inverted indices are also sorted based on a universal order and using of the index position of the last overlap in the prefix of any two records, a maximum possible overlap value is calculated. This helps in pruning records that are false positive and thereby reduces the number of candidate records to a large extent. This is an extension of the prefix filtering principle which is based on the intuition that if two canonicalized documents show similarity then it follows that some fragments of the two documents intersect. This has been implemented in the work by [5].

The empirical evaluations in this work showed that the proposed ppjoin+ algorithm outperformed the ppjoin algorithm which combines prefix filtering and positional filtering techniques also proposed in the same work, as well as its contemporary approach AllPairs. The experiments were performed on several real world datasets like DBLP, DBLP-3GRAM, TREC-4GRAM, TREC-Shingle, and the performance was compared with SOTA approaches that use prefix filtering principle. The results showed that the reduction in execution time in the proposed ppjoin+ was greater for smaller values of similarity threshold.

One of the positive aspects of this study is that the work also proved generality of its applications by extending their solution to other similarity and distance measures. Another point to note is that, this work adds to the growing corpus of research where exact computations and not approximation is used to identify near duplicates. The work proposed in this paper successfully improved the recall in near duplicate web page detection when applied on similarity join in web pages represented by shingles.

A disadvantage of this approach is that, it has a significant dependence on the similarity threshold. The reduction in execution time of this approach reduces as the similarity threshold increases. This is due to the increasing size of inverted indices when the similarity threshold is lower. Therefore in such a scenario, approaches with suffix filtering seem to outperform the other contemporary approaches.

As the similarity threshold increases closer to 1.0, the speed-up achieved seem to lessen. This approach is also tailored strictly to the domain of document similarity computation. The similarity between this work and our proposed solution is that both the works are built based on using pruning as a principle technique. In this work by [21], the Jaccard similarity is used with an upperbound threshold given by $sim(x, y) \geq \frac{t}{1+1} \cdot (|x| + |y|)$.

Jaccard similarity differs from cosine similarity measure in that, Jaccard is influenced by the length of documents being compared. Whereas, cosine similarity aims to compare the direction in which the vector representations of the two objects are aligned. Jaccard measure is also insensitive to repeation of features whereas cosine similarity is affected by terms that repeat in a document. However, cosine similarity is a better choice of measure when the vectors are more high dimensional as well as sparse. In our proposed solution, the upperbound threshold is computed iteratively based on the current k^{th} value of overlap and using the minimum() heuristic, the data is pruned for false positives.

Another similarity is the usage of sorting. The records are canonicalized using the document frequency ordering of the tokens. The tokens in the inverted index are sorted in ascending order of document frequency so that the tokens that are rare appear first, leading to small candidate set and hence speeding up execution time. Similarly, in our solution, the records are sorted based on the cardinality of their associated sets so that the more likely pair of top k records are identified faster and further causes the pruning process to accelerate. We shall see in further sections how randomizing the order can lead to sub-optimal results. However, the proposed solution in [21] largely uses a pre-determined value of similarity to build the threshold function for pruning. This makes this approach not directly suitable in our case where the requirement is to retrieve the k top *most* similar entities. Since the similarity measure here is the Overlap coefficient, it is not possible to provide a value of Overlap as threshold especially when the datasets are large in volume.

Chapter 4

Approach

As stated earlier, the goal of this work was to design a method that can efficiently identify the 'k' top most similar entities. The motivation for an efficient solution to identify the top-most similar entities can be attributed to the quadratic nature of this research problem's complexity. Given the wide applications of our research question, this work was motivated by the need to view this problem from the perspective of working on the complexity of the problem space rather than adding more computational resources to optimize the process of identifying the topk similar items. It is worthy to note here that, in this study the focus is on building an in-memory based implementation approach.

4.1 Problem statement

Most of the application areas of top-k retrieval have the advantage of using a similarity threshold in-order to continuously filter the pairs that do not meet the criteria.

And so, some of the strategies are centered around using a similarity threshold and incorporating it in different ways to solve the problem of scaling and optimizing top-k. This can be seen in the domains of document similarity detection, near duplicate detection, alternate query formulations, to name a few.

Our context however is in the application of topk in the biomedical domain. Specifically, our motivation, as discussed earlier comes from the proposition that using frequently co-occurring mutations of different genes, with the aim of finding potential biomarkers or patterns that can help in targeted treatments, disease prevention and/or better patient care management. One of the disadvantages in our context is that a threshold cannot be known apriori and therefore, we adapt to find solutions to optimize the process of finding top-k. Figure 4.1 shows a pairwise co-occurrence matrix of a small subset of oncogenic mutations.

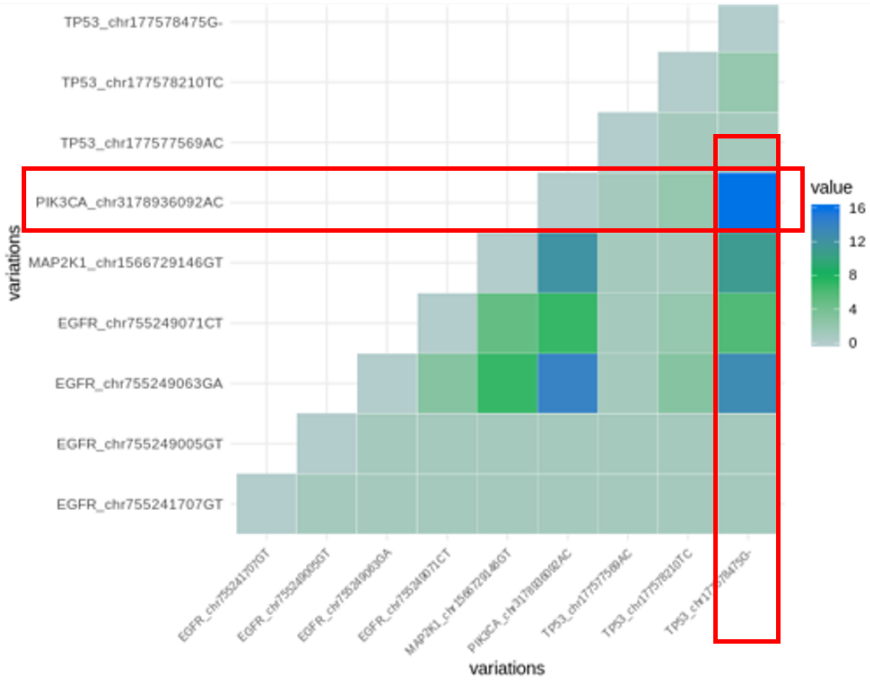


Figure 4.1: Example of pairwise co-occurrence matrix of mutations, which shows that the pair of mutations on genes PIK3CA and TP53 are the most frequently co-occurring, as highlighted.

Typically, genomic datasets are large in size, and therefore, scale and complexity of finding co-occurring mutations is a challenge. For instance, the UKBiobank data for dementia disease consisted of 46,000 unique mutations in the sequencing data that was taken from across 1,729 patients. For pairwise co-occurrences, we are already looking at a worst-case search space complexity of $(46,000)^2$. Furthermore, the idea of co-occurring mutations is well-beyond just pairwise occurrences. An exploration for dyads, triads, and even tetrads could potentially yield interesting results and serve as starting points for deeper studies. Nonetheless, within the confines of this study, we will only look at pairwise similarities. A brute force approach to solve this problem implies sequentially processing every pair of mutations, retrieving their similarity scores and then extract top-k pairs as explained in Figure 4.2. The motivation and relevance of this research comes from the need to devise an approach that can optimise the problem of top-k co-occurring mutations when a similarity threshold is not known. This solution will help in acting as an optimal first step in the study of co-occurring mutations in genomic studies more efficiently in terms of time complexity by tackling the problem complexity.

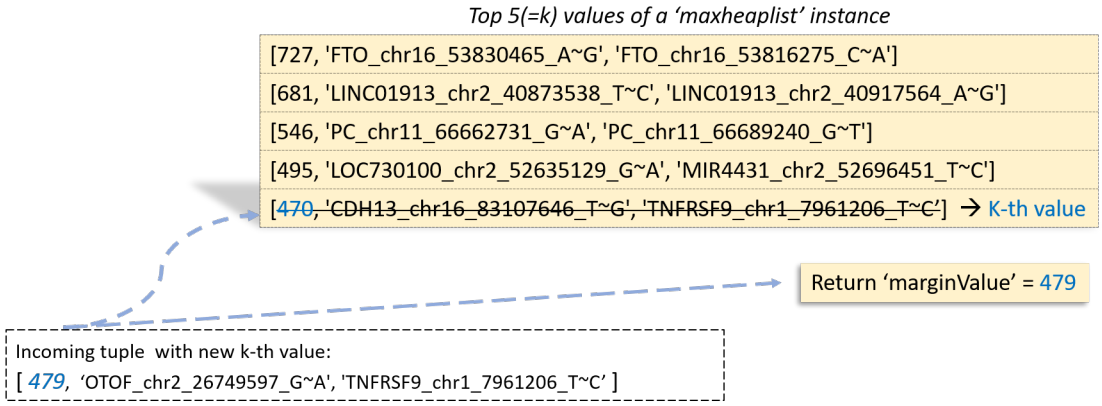


Figure 4.2: Example scenario of maxheaplist update procedure that takes place in the 'checkheap' function call. The current top k^{th} pair in the topklist is replaced when an incoming pair of mutations has a higher overlap value, 479 in this case. This new overlap value of top k^{th} pair is then returned as 'marginValue' to serve as the updated upperbound.

A formal definition of the problem is as follows: Given a dataset D with, X number of unique entities each represented in row entries of D, having a column that is the set of all patients associated with each entity, where entity $x_i \subset X$ is associated with a set $s_i \subset S$, entity $x_j \subset X$ is associated with a set $s_j \subset S$, find the top-k similar pair of entities (x_i, x_j) where, S in our case is, the set of all patients in the cohort and $k \in \mathbb{Z}$ such that $\{1 \leq k \leq (\frac{X^2-X}{2})\}$, similarity is given by:

$$sim(x_i, x_j) = s_i \cap s_j$$

4.2 Proposed solution

The aim of this study is to tackle the problem of polynomial runtime by devising an approach that reduces the complexity of the problem in order to optimize the process and also achieve high recall in the results. The hypothesis made in this study is, by using an upper bound as threshold to iteratively prune the search space , retaining high level accuracy as well as reducing complexity can bring down the time required to identify the top-k similar entities. This study also aims to investigate the impact of change in input size , change in the value of k on the execution time. Another aim of this research is to understand the impact of size and k value on contemporary approaches to better understand the computational nuances of the top-k problem.

4.3 General heuristics

In this subsection, we will look at some of the heuristics based on which the proposed solution was based on and built. These cognitive strategies were built to be applied irrespective of the format in which data is used. They are used as foundations around which the proposed solution and the pipeline to extract top-k similar entities are built. These heuristics are explained below, followed by pseudocodes.

4.3.1 Monotonic property of min()

A function on real numbers is referred to as monotonic if it is either entirely increasing or entirely decreasing. For our problem, the Minimum() gives the Upperbound for overlap measure between any two non-empty sets. Given two non-empty sets A, B with cardinalities, $|A|$, $|B|$, respectively, $|A| > |B|$ and a fixed value of $|A|$, $minimum(|A|, |B|)$ monotonically increases (decreases) with $|B|$. The following Figure 4.3 helps in understanding this better. As seen in the graph, consider the min() being applied in the setting of two sets. As the cardinality of set B decreases, the maximum possible overlap between the sets given by $minimum(|A|, |B|)$ decreases monotonically with B. This is shown using the green line in the graph.

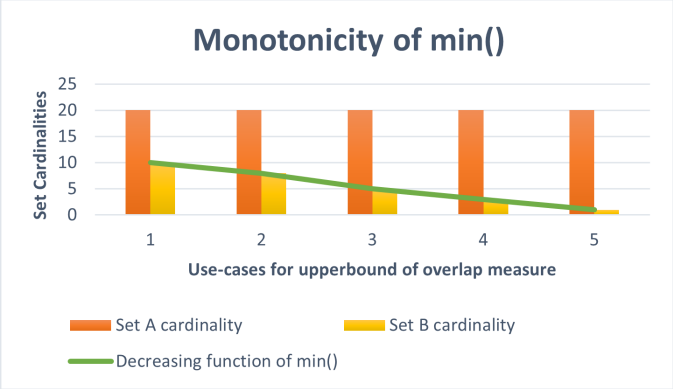


Figure 4.3: Example graph to show the monotonicity property of the min()

Therefore, we can say that when mutations are sorted in decreasing orders of their associated cardinalities, the maximum possible overlap coefficients monotonically decreases.

4.3.2 Upperbound

In order to prune the search space, the proposed solution uses the minimum property as upperbound. This is done in-order to reduce the number of false-positive computations during the process of generating the top-k similar entities, thereby guaranteeing lesser computation time and higher efficiency. Along with this background, we know

that the 'interestingness' measure used in our context is the overlap coefficient. By logic we know that, the overlap operation is bound by the lesser valued item, i.e., the overlap coefficient between two non-empty sets A and B has an upper limit which is given by the following:

$$upperbound(Overlap(A, B)) = min(|A|, |B|) \ldots\ldots\ldots\ldots\ldots\ldots \textcircled{1}$$

where $|A|$ represents the cardinality of set A and $|B|$ represents the cardinality of set B. Sets, in our context, refers to the mutations and the contents of each set is formed by the unique patients in which the particular mutation was observed, for that specific set of patient biopsies. In the following example, the mutation `'ALK_chr2_29445475_-~C'` was found to be in 4 patients and the set representation is as shown.

$$\texttt{ALK\_chr2\_29445475\_-\~{}C} = \Big[296833, 664662, 998317, 1189366\Big]$$

At any given point in time during the operation, the Overlap value of the current top-kth pair is considered as the `'threshold_value'` for that iteration. Therefore, the above mutation can have a maximum Overlap coefficient value of 4 with any other mutation having a non-zero cardinality. Using this condition, while searching the space of mutations, given a current `'threshold_value'` for the Overlap coefficient, only mutations that have an associated cardinality that is at least equal to the `'threshold_value'` can only get selected for further processing. All other mutations with cardinalities less than this `'threshold_value'` are incrementally pruned from the search space thereby reducing the number of pairs to be processed further. Therefore, if any mutation 'm' is associated with a cardinality 'c', if $c <$ `'threshold_value'`, then the mutation 'm' is pruned from the search space.

Formally, when an upperbound threshold of 'u' is applied on a column say 'c', at any given point in time, if r_i is a record with column 'c' value being v_i, in the input collection of records R, having a total of 'n' number of records, input collection of records R is given by the following expression where, $\{1 \leq i \leq n\}$.

$$\Big\{ \forall r_i \in R \mid v_i \geq u \Big\}$$

Using this property, the proposed solution reduces the number of false-positive computations incrementally. All the pairs of mutations that do not fulfill this criteria need not be investigated. This provides the required pruning effect of the search space, thereby building an overall optimal solution to the problem of top-k. Therefore, it can be said that the proposed solution uses a greedy approach, as the pruning threshold is locally updated, to find the top k similar entities without needing any similarity threshold value in advance.

4.3.3 Heuristic sorting and filtering

Size-based sorting: We know from [5] that sorting tokens in decreasing order of document frequencies can expedite the process of similarity join. In order to further reduce the time complexity, a heuristic based strategy is to order the items in decreasing order of their cardinalities. In doing so, with decreasing cardinalities, the pairs with highest values of Overlap get identified during the early iterations. Following the monotonic property of min( ), we can say that the Overlap values decrease with decreasing cardinality and therefore, reducing the chances of investigating the pair of entities that potentially could belong in the top k list. Therefore, the input mutations are ordered in decreasing order of their cardinalities [27].

Size-based filtering: In the Overlap operation , the minimum value of overlap that can occur is 1. As a pre-processing step, we remove the mutations that occur in a single patient. This is because such a mutation can only have a maximum overlap value of 1 and this can occur only with mutations occurring in that very patient only, Semantically, this does not provide any meaningful information about the cohort of patients being studied. Therefore, our data is pre-processed to contain only mutations that have a minimum individual occurrence in at least two unique patients from the cohort. Therefore, the following can be said about the range of Overlap coefficient O in the sample:

For a set of items S, where each $s \subset S$, is associated with a finite numbered set with s_{i} having the least cardinality of m_{i} and set s_{j} having the highest cardinality of m_{j}, the pairwise overlap coefficient over the set of items S has the following range properties:

$$\Big\{\, 2 \leq 0 \leq m_{\mathrm{j}} \Big\} \ldots\ldots\ldots\ldots\ldots\ldots \textcircled{2}$$

Combinations: Similarity between two entities is symmetric in nature. So is the case with the intersection operation. This implies that, given two sets A and B, intersection(A,B) = intersection(B,A). By using this property, it is possible to reduce the search space further by considering non-repeating pairwise combinations of all the unique mutations from the input [27]. Therefore, the input is processed to identify all possible unique pairwise combinations of the mutations to improve computation time. The presence of this operation in our approach provides the opportunity for this algorithm to be used in the context of identifying not just pairwise but polyads of co-occurring mutations/entities. For pairwise, where N is the number of unique mutations,

$$\binom{n}{2}$$

4.3.4 Data representation

A popular method of representing data in information retrieval applications is the term-document matrix , a vector space model of data introduced by them. In this method, every document is represented by rows of a matrix and the collection of words are represented using columns. The words that are present in a document are identified by a value of 1 and absence of a word is denoted by a value 0. The resultant matrix is known as a characteristic matrix. The method has been used in [17], where it is slightly improved by limiting the columns set to only shingles of words to reduce the computation time as it leads to a reduced number of pairwise comparisons. The same data representation idea can be extended to our context where a patient is represented using rows and mutations can be represented using columns. A patient in row p having a mutation in column m, will give the cell value in a matrix C,

$$C\ (\ p, m\ ) = 1\ \{0\ ,\ otherwise\}$$

The Table 4.1 serves as an example of a characteristic matrix.

In our context, the nature of the datasets is that the number of mutations is higher in number compared to the number of patients much like the scale of document count vs word count. Another disadvantage is the eventual sparsity of such an adjacency matrix. Since the sequencing data for a single patient can be completely different from another patient, depending on the type of biopsy, the patients could have very

Patient ID	PIK3CA_chr3_178936092_A~C	TP53_chr17_7578475_G	MAP2K1_chr15_66729146_G~T
17581	0	1	0
170471	0	0	0
192872	1	0	1
264735	0	1	1

Table 4.1: In the above characteristic matrix is 4x3 in dimension and, values C (0,0) = 0 represents 'absence' and, C (0,1) = 1 represents 'presence' where rows are in 0,1,2,3 and columns in 0,1,2.

few mutations in common. This is especially the case when the mutation counts are exponentially higher when compared to the number of patients. This implies higher computational time as well as higher storage storage space requirements. Furthermore, unlike in the domain of text mining and document similarity computation, semantically, the absence of specific mutations in patients does not imply higher similarity.

We have previously seen the different possible cases in binary vectors in section 2.1.1. In other words, in our use-case of overlap coefficient, we are only interested in one of the possible cases of 'S11' (refer section 2.1.1) i.e., the case where value is 1 or represents presence in both vectors. This way of representing can prove to be disadvantageous in large datasets, as it can lead to a highly sparse matrix, proving to be expensive computationally as well as for storage. Due to the above discussed scenarios, a different representation will be used in this work of research. Since we are only interested in 'presence' information, it is more favourable to store only information about the mutations that are known to be present in a patient. This helps in lowering the computation time [17] and also requiring reduced storage space for the operation. To be more precise about the tailored representation, since we are interested in co-occurring mutations, we store mutations (entities) and the unique patients in which they are known present. Table 4.2 shows the data representation used in our approach.

Mutation ID	PatientList	Count
CDK6_chr7_92355106_-~A	[306631, 1110078, 1124004, 1133311, 2760928, 2763499]	6
ALK_chr2_29445475_-~C	[296833, 664662, 998317, 1189366]	4
TP53_chr17_7578225_-~A	[344754, 664662, 792708]	3
APC_chr5_112175640_-~A	[2725809, 2763499]	2

Table 4.2: The above table shows the raw format of data input used in this research.Every mutation is associated with a 'PatientList' which is list of patients in which the mutation was observed. The column 'Count' is the cardinality value of the respective 'PatientList'

The number of patients that each mutation is observed to be in , is stored in the column 'Count' which, as we shall see in the further sections, will be used in the proposed solution to prune the search space. This value is basically the cardinality of the set of patients stored in the column 'PatientList'. From this point forward in this book, we shall refer to the column 'Count' as the "cardinality associated with the" respective mutation and the column 'PatientList' as the "set of patients associated with the" respective mutation.

4.4 Pseudocode

Pseudocodes are useful in conveying the overall structure of an algorithm clearly and succinctly. A pseudocode is a set of instructions written using the most expressive language in order to avoid the idiosyncrasies of any particular programming language from obscuring the essence of the concept being conveyed. The heuristics discussed in the previous section were integrated into constructing the algorithm to efficiently identify top k similar entities. The proposed solution uses two function routines to accomplish this task. The main pseudocode aims to retrieve the top k similar entities by taking the input relation and displaying the top k results. The solution uses the algorithm 2, to iteratively examine the pair of mutations, and determine if the Overlap coefficient computed is high enough to be selected into the top k list.

We will now look at the pseudocodes for both the routines in the following sections.

```
   Data: SD : input relation; SD.index : Unique mutations used as index to
         SD; SD.PatientList: Set of patients associated with each mutation;
         SD.cnt : cardinality associated with that row; k: user input value;
   Result: Top k similar entities
 1 curr_inter ← 0;
 2 marginValue ← 0;
 3 initialize maxheaplist;
 4 for 2-combinations of SD.index el(index1,index2) do
 5 |   if el is in current SD.index then
 6 |   |   curr_inter ← Overlap(SD.index1.PatientList,SD.index2.PatientList);
 7 |   |   marginValue ← checkheap(SD,curr_inter,k,el);
 8 |   |   if marginValue is not equal to 0 then
 9 |   |   |   Drop all indices from SD where cardinality is less than
   |   |   |    marginValue;
10 |   |   end
11 |   end
12 end
13 Retrieve top k similar items from maxheaplist;
```

Algorithm 1: To generate top-k similar entities

The algorithm 1 takes as input the set of all mutations in a relation data structure. The unique mutations are used as an index in the relation. Each mutation index is associated with two columns, one of them contains a list of patients that the mutation is observed in and the second column contains the cardinality or the count of patients present in this list. The algorithm also takes as input the value of k. The lines from 1-3 are used to declare and initialise the variables. The variable 'maxheaplist' is used to store the list containing the Overlap coefficient value of different pairs in a 3-item list. The unique pairwise combinations of mutations are extracted in line 4, and serve as input to the core operation of the algorithm. The lines from 5-9 , iterate over each of these combinations, to calculate the co-occurrence or intersection between their associated patient lists. This value, called 'curr_inter', is then inspected in the 'checkheap' routine if it belongs in the top k list. The response from 'checkheap' then determines whether or not the input data can be pruned further.

Any value of the variable 'marginValue' that is greater 0, results in a pruning mecha-

nism that is implemented at line 9. The pruning here is an operation of removing all the mutations that do not have the minimum cardinality of the associated patients list, in order to potentially belong in the top k list. All rows identified by mutations, where the number of patients in which the mutation is found is less than the threshold for that iteration, is dropped from the input relation.

The ensuing section explains the subroutine 'checkheap' to understand how each pair of mutations are investigated. The pseudocode for this routine is presented in the figure below. The algorithm 2 takes as input the input relation, the current Overlap coefficient value i.e. the intersection value of the pair of mutations being examined, the input value of k, and the pair of mutations. The output of this algorithm is an integer value that determines the next step to be taken in algorithm 1.

```
   Data: SD : input relation; curr_inter; k ;maxheaplist; el (index1,index2)
   Result: A value 'marginValue' that is used to determine if the pair of
           entities belong in topk list
 1 if length of maxheaplist is less than k then
 2 |   tuple ← [curr_inter, index1,index2 ];
 3 |   maxheaplist ← insert tuple;
 4 |   return 0;
 5 else
 6 |   lowestvalueinheap ← maxheaplist[k][0];
 7 |   if (curr_inter >= lowestvalueinheap) then
 8 |   |   tuple ← [curr_inter, index1,index2 ];
 9 |   |   maxheaplist ← append tuple;
10 |   |   return lowestvalueinheap;
11 |   else
12 |   |   return lowestvalueinheap;
13 |   end
14 end
```

Algorithm 2: To verify if given pair of entities belong in the top k list

The proposed solution stores the first k iterations into the 'maxheaplist'.The premise for this being that the input relation already has mutations sorted in the descending order of their cardinalities.Therefore, we hypothesize that the true-positive outcomes are likely to be retrieved in the initial iterations. During these first k iterations, no change to the input relation takes place. In order to ensure this, the return value for this scenario is set to zero. Refer to lines 1-4 in algorithm 2.

From the k+1th iteration onwards, each pair of mutations and their associated Overlap coefficient value is compared with the kth coefficient value in the heaplist in the lines from 7-12 of the algorithm 2's pseudocode.

If the current overlap coefficient value is greater than or equal to the kth value, the current pair of mutations and their associated value of overlap coefficient, is inserted into the heap accordingly. In the scenario when the current overlap value is greater than the kth value, upon insertion of this current pair into the heap list, the list instance could now be different from before. This means the kth value in the list may or may not be different.Refer the Figure 4.2 for a diagrammatic representation of this use-case. Therefore, the next step in the sequence of operations is to retrieve the current kth value in the list. This value is set as the return value of this routine and is captured by the variable 'marginValue' in algorithm 1. This value is also referred to as the threshold value for the current iteration. With a new kth largest value in the updated heaplist, we know from 4.3.2 that in-order for remaining mutations in the input relation to qualify for the future iterations, they need to have an associated value of cardinality that is equal to or greater than the kth Overlap coefficient value in the heaplist. Consequently, the 'marginValue' variable is then used to prune the search space by dropping all rows of mutations with cardinalities less than 'marginValue' in the line 9 of algorithm 1.

If the kth largest value in the heaplist is not greater than or equal to the current value of overlap, the routine continues to return the kth largest value in the heap list for the pruning process to continue for the respective iteration. The process continues until the unique number of pairs of mutations are exhausted at which point the loop ends. The top-k pairs of similar entities can now be retrieved from the 'maxheaplist'. The overall flow of execution in the proposed solution is summarized diagrammatically in the flow diagram in Figure 4.4.

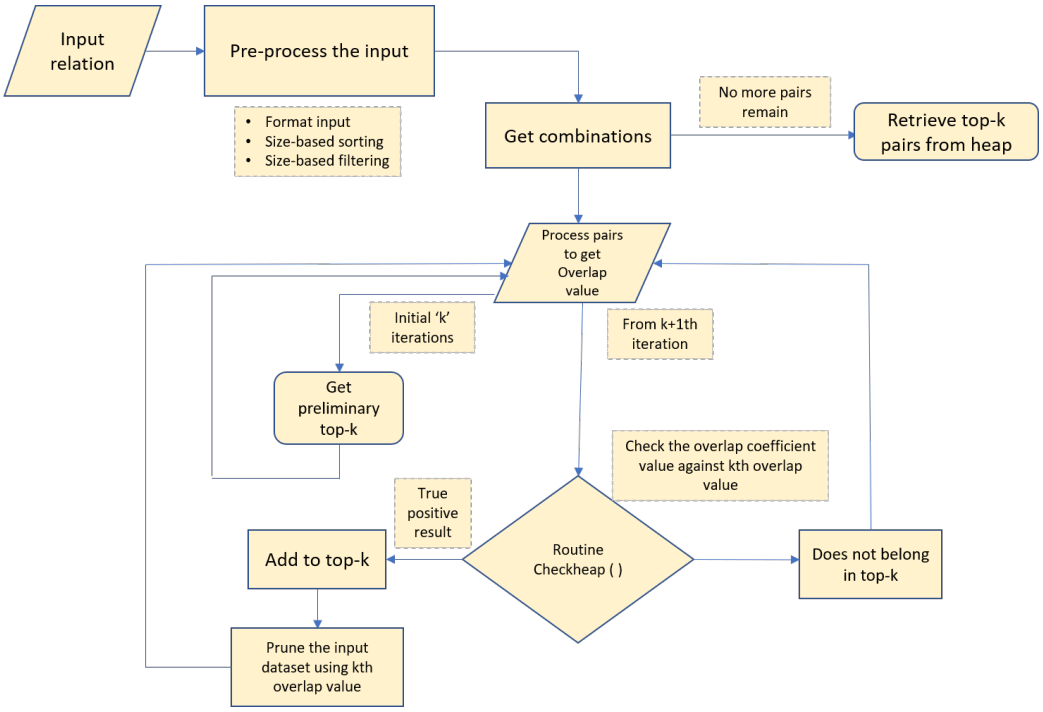


Figure 4.4: Diagrammatic representation of the flow of execution in the proposed approach. Once the input space is reduced to retain only unique pairwise combinations, the proposed approach continues to process the pairs to find the overlap. The preliminary k pairs are stored to get the k^{the} value as upperbound. This upperbound is applied to the input data if new pair has a value grater than the upperbound. This way the false positive pairs are incrementally pruned.

4.5 Runtime analysis

In order to theoretically interpret the behaviour of the algorithm, time complexity analysis is performed. In this section, we shall look at the worst case upperbound of the proposed solution. The proposed solution, as seen in the previous sections, consists of two main functions: The main function presented in algorithm 1, and

the sub-routine presented in algorithm 2. The bottleneck in algorithm 1 comes from retrieving non-repeating pairwise combinations of the mutations. We know that the formula for this is given by the following, where n represents the number of items and k represents the number of items chosen at a time:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

We also know that the total number of combinations that can be formed for all $0 <= k <= n$, is given by 2^n. This set consists of all possible subsets of 'n' including the empty set and n itself. In our context however, we are looking at pairwise co-occurrences, and therefore, the value of k remains a constant, i.e. :

$$k = 2$$

We also know that in biomedical application domains like ours, the number of mutations ($=n$) is always sufficiently large enough to say that,

$$k \leq (n-k)$$

Consider the example where n=10, k=3. Then,

$$\binom{n}{k} = \frac{10!}{7\,!\,3\,!}$$

$$= \frac{10 * 9 * 8 * 7!}{7\,!\,3\,!}$$

Let us say for large values of n, the previous equation can be re-written as:

$$\binom{n}{k} = \frac{10 * 10 * 10}{3\,!}$$

$$= \frac{10^3}{3\,!}$$

We know that k=3, n=10. Therefore,

$$\binom{n}{k} = \frac{n^k}{k\,!}$$

The worst case complexity of retrieving non-repeating combinations lies in:

$$o\,(\,n^k\,) \text{ when, } \; k \leq (\,n - k\,)$$

Since the study in this research is limited to pairwise combinations, the value of k is 2 as mentioned earlier. Therefore, the overall worst case complexity is still quadratic similar to the naive approaches that process $n(n-1)$ different pairs. However, in our case, the worst case holds only when the associated cardinalities of all mutations are equal. One of the ways to reduce the runtime behaviour is by reducing the number of pairs to be processed and that is achieved by the help of the upperbound condition that iteratively updates the upperbound condition to prune the input space.

In algorithm 2, insertion into list operation can be done in constant time and therefore, the expense comes from the sorting operation performed at the beginning of the routine to ensure that the tuples are ordered in descending order of the Overlap coefficient. The sort operation implemented in native Python 3.7 has a worst case complexity that is in logarithmic class, $nlogn$.

4.6 Summary of the chapter

This chapter presented a comprehensive explanation of the proposed solution. The chapter was segmented into the different parts to cover the heuristics, properties and data representations used in building the solution. The proposed solution was presented in the form of pseudocodes that help in understanding the flow of execution of the approach which makes it easier to be implemented in a programming language agnostic method. An analysis of the runtime behavior was also presented to understand how our approach differs in its nature compared to the naive approach.

Chapter 5

Implementation

This chapter explains the implementation of the proposed solution presented in the previous chapter. The pseudocode of the proposed approach has been implemented using the programming language Python in version 3.7

5.1 Input data source

The data required for this research work was obtained using the next generation sequencing, a DNA sequencing methodology in genomic research. To get a basic idea of NGS, from [20], we can understand that NGS platforms are used to perform parallel sequencing of many small DNA fragments. Each base in the human genome is sequenced multiple times, thus providing highly accurate data and information about unseen mutations in the DNA. The NGS process can be applied to either complete genome sequences of particular areas, like the a whole exome or even a few individual genes.

The data we received consists of information about the different genomic mutations present in the biopsy results of several patients of a specific disease. The raw data was in the form of a .csv file with each row containing patient identification number and the mutation present. The mutations were identified and named using a specific format as per the project guidelines. The data was stored in the form of a table with two columns, one containing the patient identifier and the second column with the mutation observed to be present in the patient. Every patient is observed to have multiple mutations of various genes and this is stored in the form of multiple rows corresponding to a single patient. The Table 5.1 shows the format of data that was received and further used as input our pre-processing pipeline.

Patient id	Mutation
2239685	TP73_chr1_3570883_C~T
4265751	TP73_chr1_3570883_C~T
5201126	TP73_chr1_3570883_C~T
.....	.....
2526011	TP73_chr1_3574756_T~C

Table 5.1: Raw format in which the data was received. The data is in long format where patient id is repeated as many times as the number of mutations that were observed in each patient.

As mentioned earlier, to optimize the process of finding top-k similar entities, the proposed solution works in the direction of reducing the problem complexity by pruning the search space. The above data is pre-processed in-order to transform it into a format that is suitable to be used as input into the proposed solution. The proposed solution takes as input, the data in the form of a relation.The main operation is based on the 3 essential columns listed below.

1. Mutations
2. Patient list
3. Count of patients

Inorder to convert the input data in the format shown in the above figure, the preprocessing pipeline has been implemented in R Studio.

To convert the long form of data into the wide format, reshape2 library's *'dcast'* functionality has been used. This results in a binary representation of the data where each patient is represented by a row, with presence and absence of a particular mutation is represented by either 1 or 0 respectively. But, this form of representation as already discussed requires more storage space and also the triviality of the absence information of mutations leads to the format where only the presence information is stored. Therefore, from the binary representation, the data is combined column-wise based on mutations and aggregated to over rows (patients) where the value is equivalent to presence or 1. This is achieved using the native *aggregate()* function in R. Applying the aggregate function results in a column with comma separated list of patients associated with each mutation.

The cardinality of each set of patients is important for the implementation of the proposed approach and to accomplish this, *strsplit()* functionality has been applied to each row of the 'PatientList' column and number of items in the set are counted and stored in a new column 'Count'. The table in Figure 5.1 represents the final result of applying the aforementioned data manipulation pipeline in order to convert the raw data into a format suitable for the proposed approach.

Mutation	PatientList	Count
CDK6_chr7_92355106_-~A	[306631, 1110078, 1124004, 1133311, 2760928, 2763499]	6
ALK_chr2_29445475_-~C	[296833, 664662, 998317, 1189366]	4

Figure 5.1: Example showing the input format of data used in the proposed approach. The data here is manipulated by grouping based on the 'Mutation' column.

This data is further filtered to remove those mutations that occur in less than 2 patients. The data is also sorted in the descending order of the 'Count' column so that the most highly frequent mutations are processed during the first iterations. After applying the data manipulation pipeline, the files are exported locally in the .csv format and later imported as input to the proposed approach. The Figure 5.2 depicts the main lines of code used in this initial processing step.

```
# APPLY RESHAPE2 LIBRARY'S DCAST FUNCTION TO CREATE A WIDE-FORMAT DATA
dcastfordata<-dcast(SourceData, patientList ~ variation,
fun.aggregate = length)
# APPLY AGGREGATE FUNCTION TO GROUP THE DATA COLUMN-WISE AND STORE
PATIENT ROWS WHERE VALUE =1
source_dataset<-aggregate(patientList ~ Mutation, subset(cbind(
patientList = as.integer(dcastfordata$patientList),
stack(dcastfordata[-1])), values > 0), toString)
# CREATE A NEW COLUMN 'COUNT' TO INCLUDE THE NUMBER OF PATIENTS
source_dataset$Count<- sapply(strsplit(as.character(source_dataset
$patientList), ","), length)
```

Figure 5.2: Lines of code from the data manipulation process

5.2 Python libraries

The implementation of the proposed approach has been done in Python version 3.7. Jupyter notebook has been used as the choice of editor and execution environment. One of the important libraries used in the implementation is the 'combinations' package of the 'itertools' module version 8.0.2 of the Python Package Index.

The 'combinations' generator module takes an array of given length n as input and generates non-repeating combinations of length r. The 'Mutation' column is indexed into the input relation and is used as input to the combinations() module with a value of $r = 2$ for pairwise combinations.

The second important package used is the 'heapq' module. The nlargest() function of this module is used to extract the k most similar entities in the proposed solution. This function takes the list or iterable specified as input along with the value of k. Normally, this module is used for heap ordered data structures. However, in the implementation of the proposed approach, the input here is a list of tuples with 3 items, the Overlap coefficient value, and the identifiers of the associated pair of mutations. The Figure 5.3 shows an instance of how the proposed solution outputs the top 10 pair of similar entities on an execution.

```
[[426, 'TMEM132C_chr12_128892549_A~G', 'SLC15A4_chr12_129200312_A~G'],
 [414, 'TMEM132C_chr12_128892549_A~G', 'FBXL7_chr5_15737308_C~T'],
 [414, 'TMEM132C_chr12_128892549_A~G', 'ECE1_chr1_21525228_T~C'],
 [407, 'SLC15A4_chr12_129200312_A~G', 'FBXL7_chr5_15737308_C~T'],
 [406, 'SLC15A4_chr12_129200312_A~G', 'LOC100287015_chr8_5994657_G~A'],
 [405, 'TMEM132C_chr12_128892549_A~G', 'TMEM181_chr6_158944233_G~T'],
 [404, 'TMEM132C_chr12_128892549_A~G', 'LOC100287015_chr8_5994657_G~A'],
 [396, 'FBXL7_chr5_15737308_C~T', 'LOC100287015_chr8_5994657_G~A'],
 [394, 'SLC15A4_chr12_129200312_A~G', 'ECE1_chr1_21525228_T~C'],
 [390, 'FBXL7_chr5_15737308_C~T', 'ECE1_chr1_21525228_T~C']]

Execution time= 0.24733805656433105 seconds
```

Figure 5.3: **Output of execution:** Top 10 similar pairs as returned by the proposed approach, which takes an overall execution time of approximately 0.25 seconds

5.3 Baseline implementation

A baseline can be defined as any method or a "starting point" that is used as a comparison.As a baseline, the proposed approach has been compared to two other methods of efficiently retrieving top-k. The following section discusses the implementation of two state-of-the-art approaches that we will refer to as our baselines or 'contemporaries' and compare the efficiencies in terms of runtimes to evaluate the

proposed approach against the baseline approaches.

Data: input_data; input_columns;k
Result: Top k similar entities

```
1 resultArray ← [ ];
2 for non-repeating pairwise combinations of input_columns do
3 |   resultArray ←
  |     append( combination pair, # of rows where both columns have value 1);
  |
4 end
5 resultData ←
    table created from resultArray with columns = ["Var1", "Var2", "Frequency"];
    // where "var1" & "var2" columns store entity pair identifiers
6 Display 'k' largest according to column 'Frequency';
```

Algorithm 3: To retrieve top k similar entities using 'non-exhaustive brute force computation' also referred to as 'OEM' in this thesis.

The first baseline, algorithm 3, is an implementation of an improved "naive approach". As we already know, the traditional approach is brute force in nature, with a time complexity in $O(n^2)$. This is because, in this approach, to find the top most similar entities , every entity in the set is compared with every other entity to calculate the similarity value and then the top similar pairs are retrieved. This process does not use any strategy to reduce the number of computations required. And therefore, the run time and efficiency becomes a bottleneck as the size of the input increases, essentially leading to a quadratically increasing runtime.

However, we also know from [17] that similarity is a measure that is symmetric and therefore, by applying this property, the effective number of computations can be reduced to $n(n-1)/2$. Even though this approach is still brute force in nature, in that every pair of non-repeating entities are processed, it is less exhaustive in nature. Consequently, even with the same worst-case time complexity in $O(n^2)$, the runtime behaviour is lower. In the context of this work, this approach is referred to as the "optimized exhaustive method" method of computation or "baseline-I". The algorithm 3, explains the steps used in implementing this approach as one of the baseline methodologies that will be studied and analysed in this work of research.

The input data for this approach is, however, a binary representation of the various 'Mutations' found to be observed in different patients of the cohort. The Table 4.1 serves as a diagrammatic reference.

This approach has also been implemented using Python 3.7, the execution environment and editor of choice being Jupyter notebook. As with the proposed approach, the implementation of this baseline-I uses Python's itertools module for its 'combinations()' package to retrieve non-repeating combinations of the Mutations and the 'heapq' module's 'nlargest()' function to retrieve the top k similar entities.

The second baseline approach used in the context of this research is the Top-k cOsine similarity Pairs using MAx-first TrAversal method or 'TOP-MATA' algorithm [27]. TOP-MATA is implemented in the domain of association analysis to find top k strongly related item pairs. Being in the high-dimensional space is typically characterized by large scale datasets and it uses cosine similarity as the 'interestingness' measure of choice. Consequently, the properties of cosine similarity in terms of support measure has been used to build the upperbound condition. TOP-MATA processes item pairs in a max-first traversal approach. In this approach, the processing starts with the row containing the highest valued item pair. Refer to the Figure 5.4 for a visual depiction of the execution, where items are represented in the rows and columns of the matrix.

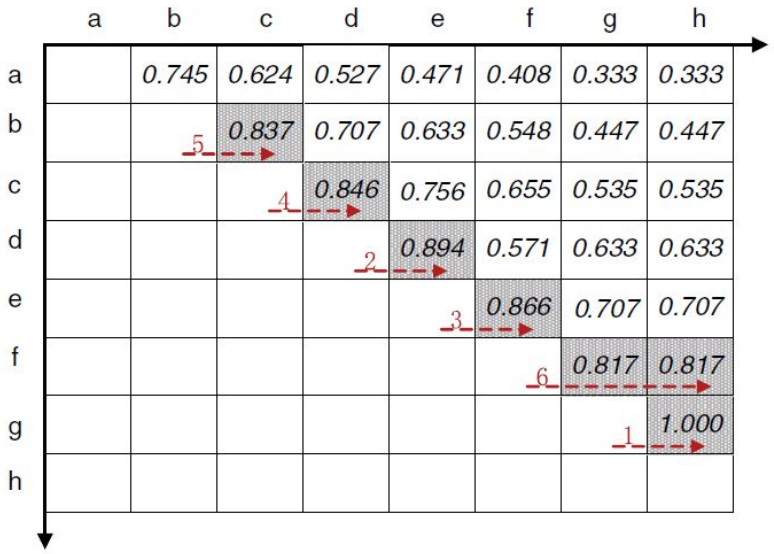


Figure 5.4: From [27] showing the max-first execution of TOP-MATA indicating the order in which the rows in the upper triangle are chosen for processing.

Since the domain of implementation of TOP-MATA is in association analysis, the items are represented using binary vectors. The sorted matrix is built using the upper cosine value of a pair given by the following equation, where X and Y represent different items.

$$upper(cos(X,Y)) = \sqrt{\frac{support(X)}{support(Y)}}$$

This is the pruning criteria / upperbound used. TOP-MATA basically uses a two-tier examination of the item-pairs in each iteration. The first step is in selecting only those item pairs that have the highest value of upper cosine in a row and in the second step, is the calculation of the actual cosine similarity of the chosen item pairs which is then checked against a minimum value of cosine similarity to determine if the item pair deserves a position in the top-k. The following are the three theorems that have been applied in the implementation of the max-first traversal strategy in TOP-MATA to scale top k computation that are demonstrated in [27].

Theorem 1 [27]: Given two items X and Y with $support(X) >= support(Y)$, $upper(cos(X,Y))$ is monotonically increasing in support(Y) and monotonically decreasing in $support(X)$.

Theorem 2 (The filtering effect) [27]: A new pair X,Y can enter the top-k list only if $upper(cos(X,Y)) >= minCos$.

Theorem 3 (The pruning effect) [27]: Given the current top-k list and its minCos, for pairs in sorted item-matrix, if $upper(cos(P[i,j])) <= minCos$, then for all $k <= i$ and $l >= j$, $upper(cos(P[k,l])) <= minCos$.

In order to implement TOP-MATA, the following adaptations were executed. In the approach presented in [27], the upperbound condition is changed using $min()$ property of sets instead of the $upper(cos(X,Y))$ property. Therefore, the sorted item-matrix has values of absolute cardinality and is constructed by arranging them in the descending order of cardinality as is followed in TOP-MATA.

With the aforementioned changes, the table in Figure 5.5 represents the sorted item-matrix M of dimensions $n \times n$, where each cell identified by row i and column j, has a value that is given by the following, where , $\{1 \leq i \leq n\}$ and $\{1 \leq j \leq n\}$:

M (row i , column j) = min (| associated cardinality of mutation at row i |, |associated cardinality of mutation at column j |)

	MIR4431_chr2_52696451_T~C	LINC02210-CRHR1_chr17_43802506_A~C	ZNF862_chr7_149534216_A~G
MIR4431_chr2_52696451_T~C	--	688	575
LINC02210-CRHR1_chr17_43802506_A~C	--	--	575
ZNF862_chr7_149534216_A~G	--	--	--

Figure 5.5: This sorted item matrix of mutations is the adaptation of TOP-MATA for our use-case. The cell values in the matrix are filled by applying the min() property to the individual frequencies of every pair of mutations that identify the respective cell.

Apart from this change in upperbound condition, the rest of the implementation follows the same procedure as in TOP-MATA. The pseudocode of TOP-MATA implemented in this work of research is shown in algorithm 4.

```
Data: input relation SD; SD.index : Unique mutations used as index
      to SD; SD.PatientList: Set of patients associated with each
      mutation; SD.cnt: cardinality associated with that row; k: user
      input value;
Result: Top k similar entities
minCos ← 0;
N ← size of SD;
k ← user input value;
for i = 0 to N do
    j ← i + 1;
    value ← min(i, j) ;
    Append( maxheap, (i,j,value));
end
maxUpper ← maxheap.Root;
while maxUpper ≥minCos do
    rowi = row index of maxheap.Root;
    rowj = column index of maxheap.Root;
    je = item index whose min value is less than rowj^th;
    for j = rowj to je − 1 do
        cos = Overlap(SD.rowi.PatientList , SD.rowj.PatientList);
        if cos > minCos then
            Add to topKPairs;
        end
    end
    if je < length(SD) then
        upper = min(SD.rowi.cnt , SD.je.cnt);
    else
        upper= min(SD.rowi.cnt, SD.rowj.cnt);
    end
    if upper > minCos then
        maxheap.Root = rowi,je,upper;
    else
        maxheap.Root=rowi,N,-1;
    end
    Heapify maxheap;
    if length(maxheap) ≥ k then
        minCos= topKPairs[k][0];
    else
        minCos=1;
    end
    maxUpper=maxheap.Root ;
end
Retrieve top k similar entities from topKPairs
```

Algorithm 4: TOP-MATA adaptation to find top k similar pairs

The implementation of TOP-MATA has been done using Python 3.7. The 'heapq' module is used to implement its 'nlargest()' function in order to extract the k most frequently co-occurring pairs of entities at the end of the procedure.

5.4 Summary of the chapter

This chapter provided a detailed explanation of how the proposed solution was implemented covering the technology that was used, the important libraries used in the implementation of the proposed solution and also provided an insight into how the state-of-the-art approaches were adapted and implemented to solve the problem statement in our domain.

Chapter 6

Experimental Evaluation

This chapter presents a study of the performances of the three different algorithms that were implemented on datasets of several different sizes. Specifically, the aim is to demonstrate the following:

1. Runtime behaviour analysis of the algorithms
2. Completeness in terms of accuracy in performance of the algorithm

In this chapter, apart from presenting the proposed approach to efficiently identify the top k similar entities, our work also aims to answer the following research questions and further present the empirical analysis of the research questions.

1. The impact of different input size on execution time;
2. The impact of different k values;
3. Number of iterations required;, and
4. the influence of cardinality distribution on execution time.

6.1 Experimental setup

All the experiments were run on a 64-bit windows 10 system with 16GB RAM, single core i7-8550U 1.8GHz processor. To convert the data into a suitable format for the different approaches, the data manipulation processes was performed using R programming. The files were then exported locally in comma separated format before using as input in the Python implementations of the approaches. The following Table 6.1 gives size characteristics of the different datasets used in the experimental evaluation.

Number of unique mutations	File size
40	132 KB
66	175 KB
100	273 KB
300	815 KB
500	1261 KB
1000	2813 KB
1500	4332 KB

Table 6.1: File size property of different inputs files used in the experimental evaluation. All the files are in the .csv format.

6.2 Metrics

In order to evaluate the completeness of our proposed algorithm, we shall compute sensitivity/recall. Recall is given by the fraction of total relevant instances that get retrieved. Within the context of our domain, the top k similar items are considered as belonging to the true positive class and the rest of the mutation pairs are regarded as true negative thereby, considering this problem in terms of binary classification. In the context of binary classification, we know that recall is also referred to as

Sensitivity or true positive rate. It is given by the number of true positives over the sum of true positives and false negatives. In our context, it is more important to identify how many of the retrieved top similar entities are 'false positives' and hence do not belong. Therefore , to evaluate the completeness of the approaches, we measure misclassification rate or error rate which is given by the number of incorrectly retrieved instances.

$$\text{Misclassification rate } = \frac{False\ positives + False\ negatives}{Total}$$

6.3 Performance study and analysis

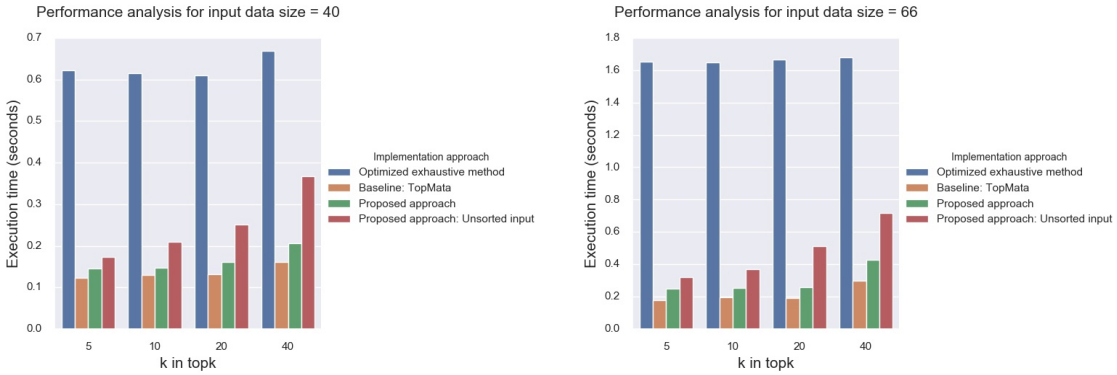
The contributions of this work of research includes the study of how the performance of the proposed approach is impacted by different values k, different sizes of input the number of iterations required for different values of input size and the influence of cardinality distribution on execution time. As part of the empirical evaluation and study of the proposed approach, various experiments were performed to study the aforementioned use-cases and the results were compared with two baseline approaches. This helps in understanding the complexity of the research question deeper and to hypothesize solutions for future work. This section is further divided into 3 subsections to present the results of the experimental evaluation, followed by a discussion these results of the 3 different experimental use-cases.

6.3.1 Run time analysis based on 'k' value and input size

In this section of the study, the execution time if used as a measure to evaluate the performance of the proposed approach when compared to the optimized exhaustive approach and the baseline methodology, TOP-MATA. The experiments are conducted for different values of k in top-k. This is done in-order to understand the impact of increasing or decreasing the k value has on the three different approaches. Further, this is also implemented for various sizes of input to analyse the dependence of execution time on input size. The above Table 6.1 gives a brief overview of the

various input volumes and the size of the input data source used in the experiments.

The graphs in Figure 6.1, show the execution time of three approaches for four different increasing values of 'k' for data input of sizes 40 and 66. This input data consists of 40 and 66 unique mutations present in x number of patients, respectively. Considering that this is an input of small volume, we observe small execution times. The proposed approach has also been executed by taking unsorted input. This implies that, the mutations in the input are not sorted based on their associated cardinalities. Since the proposed approach relies on retrieving pairs with higher values of co-occurrence in the early iteration, the performance with unsorted values of input shows higher execution time values. In both these cases shown in Figure 6.1a and Figure 6.1b, we observe that the 'optimized exhaustive method (OEM)' performs worse than other methods recording execution time of approximately 0.65 seconds for the 'k' value of 40. In this section of experiment, TOP-MATA is observed to have superior performance compared to other approaches. Another aspect observed in the behaviour of TOP-MATA is the close range of runtimes for different values of 'k'. With the increasing values of 'k', the increase in runtime is not significant enough and therefore shows scales well with 'k'. The proposed approach in this case also shows better performance compared to OEM, however, the execution time is slightly higher than TOP-MATA for all values of 'k'.



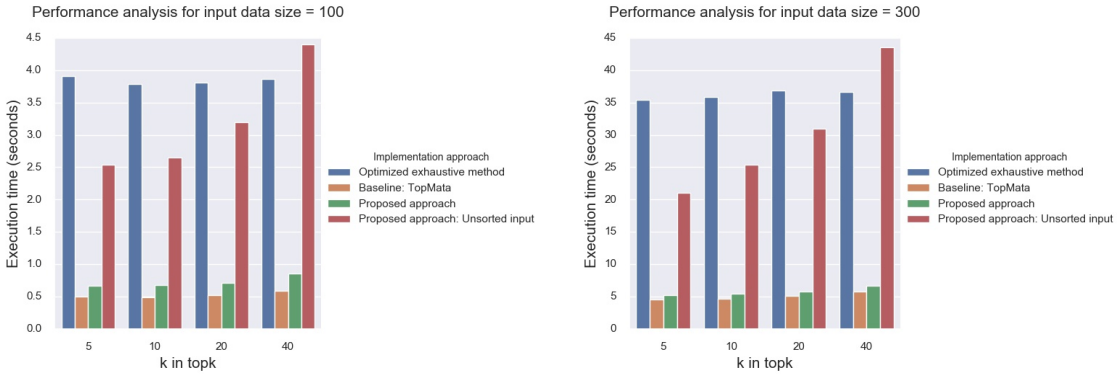
(a) Execution time for input size of 40.

(b) Execution time for input size of 66.

Figure 6.1: Performance analysis with different 'k' values for two input sizes.

The results in Figure 6.2 depicts the performance of the approaches with slightly higher volume of input data, i.e., 100 unique mutations in Figure 6.2a and 300 unique mutations in Figure 6.2b. It is interesting to note at this point that the OEM approach performs slightly better compared to the performance of the proposed

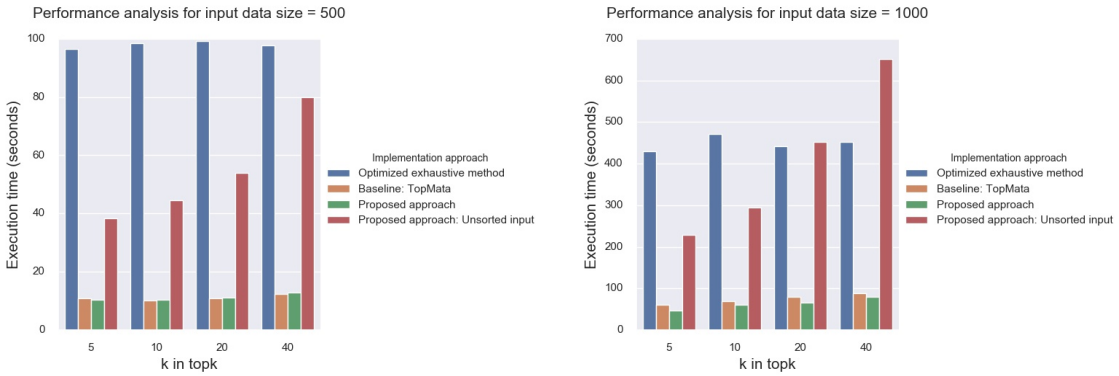
approach on unsorted input. This is owing to the 'checkheap' routine that gets called more number of times when the input is sorted in a random order. As a worst case scenario, it logically follows that, if the input is sorted in ascending order of associated cardinalities, every reduction/pruning of the search space leads to very few records from being pruned at every iteration. This is one of the reasons that leads to higher execution time and lower performance compared to the brute force approach. It can be seen in the use-case of Figure 6.2b that the proposed approach performs with runtimes that are similar to TOP-MATA.



(a) Execution time for input size of 100.

(b) Execution time for input size of 300.

Figure 6.2: Performance analysis with different 'k' values for two input sizes, 100 and 300.



(a) Execution time for input size of 500

(b) Execution time for input size of 1000

Figure 6.3: Performance analysis with different 'k' values for two input sizes.

In both Figure 6.1 and Figure 6.2, with increase in 'k' value, the increase in execution time is in small quantities in both the proposed approach as well as TOP-MATA. However, the execution time remains fairly stable in case of OEM.

Further, the performance comparisons depicted in Figure 6.3a for input data of size 500 and Figure 6.3b for input data of size 1000 show slight changes in pattern. In Figure 6.3a, it can be seen that the execution times of TOP-MATA and the proposed approach are identical, with OEM still continuing to show sub-par performances for all values of 'k'. Also, to be observed is that both the proposed approach as well as the TOP-MATA show minimal changes with increasing 'k', and are scalable in 'k'.

Depicted in Figure 6.3b, for a relatively large number of unique mutations, the proposed approach is seen to perform with the lowest execution time for all values 'k' and therefore showing highest efficiency comparatively. In all of the cases, the change in the value of 'k' shows the highest impact in execution time when the proposed approach is executed on unsorted input. However, the baseline TOP-MATA shows the most optimal performance when the size of the input is relatively smaller with lowest values of execution time across all values of 'k'. Whereas, for larger values of input i.e. the proposed approach performs better than all other, across all values of 'k'.

The overall comparison of performances with respect to different input sizes for a single value of 'k' is shown in Figure 6.4. This helps to provide a comprehensive understanding of how the different approaches are dependent on the input size and 'k' value. As seen, the proposed approach shows optimal performances for increasing sizes of input, showing better performance than TOP-MATA with increasing values of input.

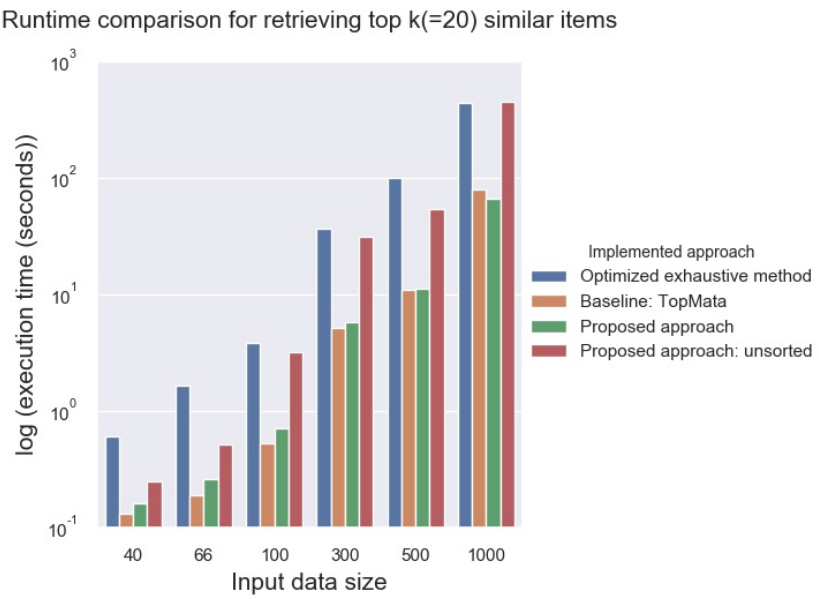


Figure 6.4: Execution time comparison for different input sizes. As the input size increases, the proposed approach shows increasingly better runtime performance, having the lowest runtime for the input size of 1000.

Since the OEM showed a timeout for an input data of size 1500, only the proposed approach and our implementation of TOP-MATA were applied. The results can be seen in the Figure 6.5.

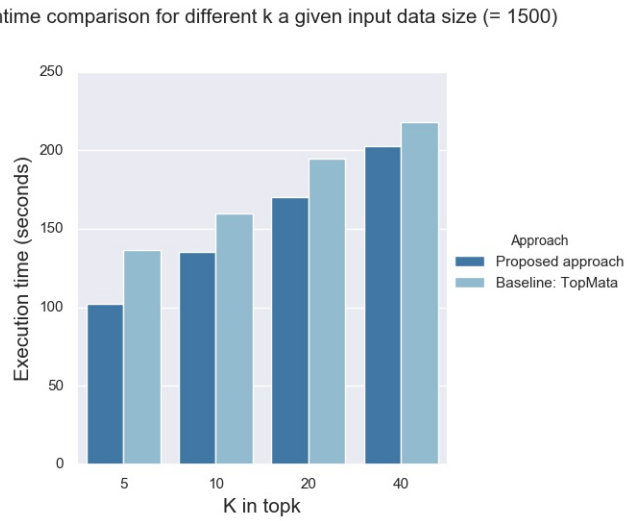


Figure 6.5: Execution time comparison for input size of 1500. It can be seen that for all values of k, the proposed approach is showing a runtime that is lower than that of the TOP-MATA implementation. However, it can also be noticed here that as the k value increases, the runtimes tend towards sameness as the percentage of difference is decreasing.

From the graph in Figure 6.5 , we can observe that the proposed approach is executes with the least runtime in this case when compared to our TOP-MATA implementation. This can be seen across all the values of k. Therefore, from our experimental evaluation, we can say that with increasing volume of input, the proposed approach is more efficient in terms of runtime with TOP-MATA consistently showing higher execution times.

6.3.2 Reducing the number of false positives

One of the strategies that TOP-MATA uses to optimize it's design is by reducing the number of false positive computations. The process is also maximized with the max-first traversal strategy. This design seems to work better when the size of the input data is small. The proposed approach however, shows sub-optimal performance with small sized input data as they complexity of calling the 'checkheap()' routine adds to the execution time. Specifically the sort operations adds to the time complexity thereby leading to lower measures of performance. However, in case of input data

of bigger size, the pruning leads to faster reduction in search space compared to max-first traversal and therefore, the proposed approach seems more suited for all the considered values of 'k' as the input data size increases.

6.4 Effect of cardinality on runtime

In this study, the impact of the cardinality distribution on runtime of the different approaches is studied. We know that each of the mutations in the input data is associated with a 'count' value which corresponds to the number of patients in which the mutation was observed to be present in the particular patient cohort. We refer to this value as the 'associated cardinality'. This associated cardinality value has an impact on execution time as the operation of calculating the intersection is directly dependent on the length of the sets being evaluated. The average case of time complexity is given by the following, where s and t are both sets.

$$O(min(len(s), len(t)))$$

Since both the proposed approach as well as our baseline implementation of TOPMATA calculate overlap coefficient as the raw value of intersection of the two sets associated with the mutations, this becomes an interesting use-case. In order to perform this experiment, two different datasets have been used with different cardinality distributions. for simplicity, the first dataset, will be referred to as 'Dataset 1'. Dataset 1 , is observed to have a lower average cardinality compared to the second dataset , referred to as 'Dataset 2'. The Figure 6.6 shows the statistical distributions of the two datasets.

count	66.000000
mean	296.409091
std	269.448438
min	7.000000
25%	80.000000
50%	201.000000
75%	463.250000
max	860.000000

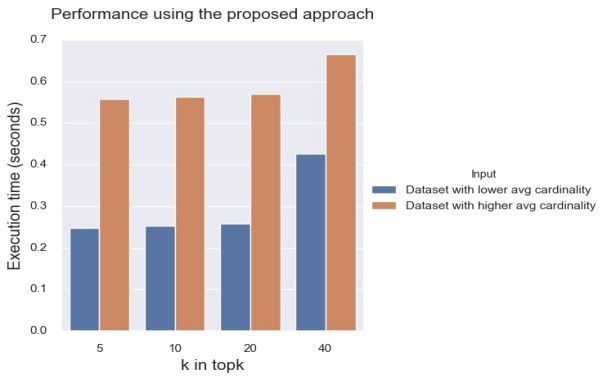
(a) Statistical properties of 'Dataset 1'

count	66.000000
mean	407.909091
std	307.344894
min	6.000000
25%	120.500000
50%	334.500000
75%	726.000000
max	859.000000

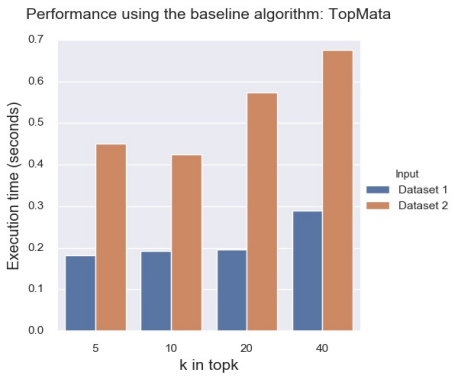
(b) Statistical properties of 'Dataset 2'

Figure 6.6: Statistical properties of the cardinality distribution in two datasets, where 'Dataset 1' has lower overall average compared to 'Dataset 2'.

In order to limit the impact of the input size on the execution times, the size of input data for this experiment has been limited to 66 for all approaches.The Figure 6.7 illustrates the execution times for selected 'k' values in two datasets when the proposed approach and our implmentaion of TOP-MATA is applied.



(a) Performance of the proposed approach



(b) Performance of TOP-MATA

Figure 6.7: **Comparison of performance** to analyse the impact of cardinality distribution on the proposed approach and our implementation of TOP-MATA. Both approaches show similar behaviour as the overall cardinalities in the data increase. However, as 'k' increases, our TOP-MATA implementation shows a sharper increase in runtime for 'Dataset 2' compared to the behaviour seen in the proposed approach

We can say that the results in Figure 6.7 are in alignment with the conjecture that the distribution of cardinality impacts the execution time in both approaches. The 'Dataset 2' with higher avg cardinality, is observed to take more time as 50% of the values of cardinalities has a maximum of 334.5 when in 'Dataset 1' the 50% percentile value of 201. However, the above Figure 6.8 shows the impact of cardinality on the

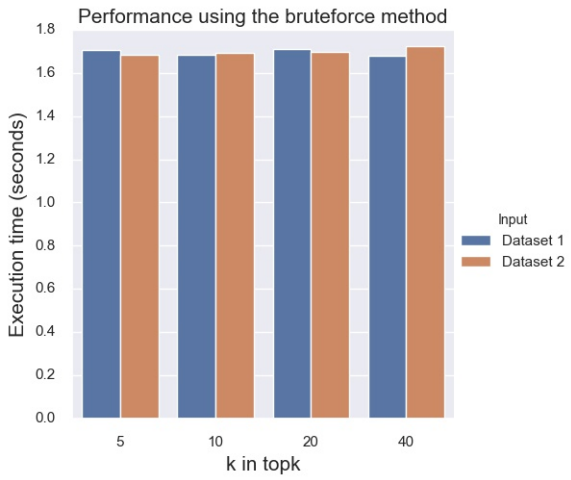
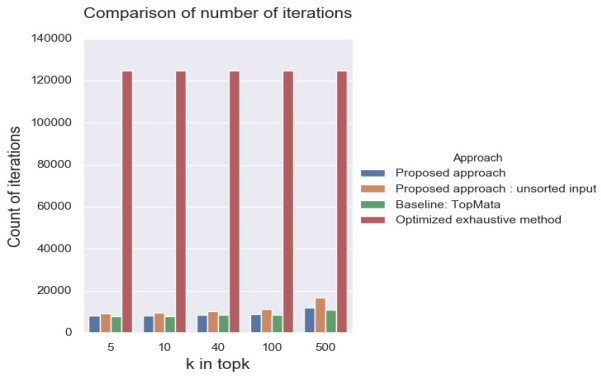


Figure 6.8: Impact of cardinality distribution on execution time in OEM. As seen OEM shows similar execution times for both datasets thereby proving to be independent of the cardinality distribution in the input dataset

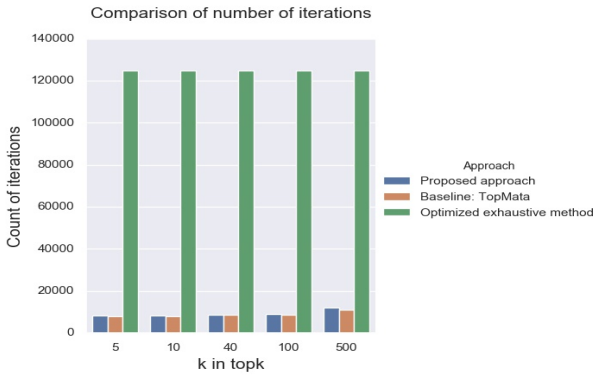
OEM, which is executed on a binary representation of the data with co-occurrence computed as a count of rows where both mutations show presence for a given patient. This implies that, the complexity is dependent on the total number of unique patients represented by the number of rows, since each row is examined for a given pair of mutations. Consequently, this leads to $O(n)$ time where n is the number of items to be summed. Therefore, the increase or decrease in number of co-occurrences does not have significant impact on execution time. In case of both datasets, the OEM shows similar execution time for all the selected values of 'k'.

6.5 Comparison of the number of iterations

The graphs Figure 6.9, shows the number of iterations required by the various approaches, in order to fetch the top k similar entities from an input data of size 500. The count of iterations is dependent on the strategy used by the different approaches in solving the problem of top k.Apart from time intensive operations, heuristically, it is clear that higher number of iterations implies a higher execution time value. Empirical evidences in Figure 6.9, show that the brute force approach OEM clearly requires significantly larger number of iterations compared to the other approaches. Considering that in OEM, other than the symmetric property of combinations, no other process is employed to reduce the number of false positive computations, it shows consistently higher number of iterations in comparison. Therefore, instead of requiring a naive $n(n-1)$ number of operations before retrieving the top k pairs, the OEM only uses: $n(n-1)/2$ operations. And as OEM does not include any strategy to reduce the number of computations, the iteration count for all the values of 'k' in a given input space, does not change.



(a) Number of iterations in different approaches including unsorted input in proposed approach



(b) Number of iterations in different approaches excluding the performance of proposed approach on unsorted input.

Figure 6.9: Comparison of the number of iterations required to fetch top k entities in an input of size(=500). Note that the number of required iterations is identical in the proposed approach and our TOP-MATA implementation and yet execution times differ.

However, we know that the search space can be further reduced by employing heuristic measures. This is evident in the number of iterations required by TOP-MATA and the proposed approach. The key operation used to evaluate the overlap coefficient in both TOP-MATA and the proposed approach is the same but the difference in time required arises from the 'checkheap' routine in the proposed approach that investigates the pairs against the current upperbound every iteration. But, it can be observed that the number of required iterations in both the proposed approach and TOP-MATA presents to be comparable.

It is interesting to note the dependence of the proposed approach on the format of input data. In section 2.3, the importance of data access was presented.The cost of random access as compared to sorted access in the context of the proposed approach can be seen in Figure 6.9a. The proposed approach on unsorted input shows suboptimal performance with respect to the iteration count in comparison to both TOP-MATA and proposed approach on sorted input. This is potentially due to the below surmised reasons:

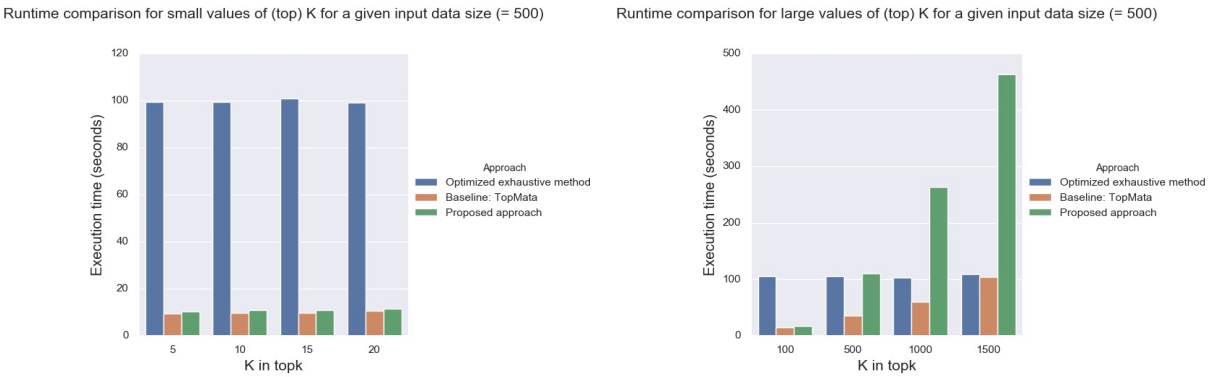
1. With unsorted input, the process of pruning the search space becomes slower due to randomly distributed records with lower associated cardinality values

2. With random distribution of records in input, it also leads to more number of insert operations into the heap and re-shuffling of the current top k pairs as records with higher associated cardinalities are processed during later iterations

The rate of increase in the number of required iterations with increasing values of k, could indicate how well the approach scales. As seen in Figure 6.9, the OEM has significantly high values of iteration count for all the selected values of k. The number of iterations however, does not change across different k values. However, in the proposed approach as well as our baseline TOP-MATA implementation, the number of iterations are affected by the 'k' value. In the proposed approach, when the k value increases by 5 times, the number of required iterations become 33.76% more. But in the case of our implementation of TOP-MATA, the number of required iterations increase by only 27.3%, for a similar increase in k value.

6.6 Performance analysis for extreme values of k

The next use-case in this study is the investigation of how the approaches perform with very high values of k as compared to lower values. Most of the applications are concerned with the low values of k compared to the relatively high input sizes.(provide paper reference) However, for the sake of comprehensive research it is important to understand how the approaches scale not just with increasing input size but also with increasing values of k. In applications related to market basket analysis, frequent itemset mining, the top k similar entities are not limited to top 5 or top 10 rather at more commercial scales. Therefore, in order to assess the generality of the proposed approach, this use-case was deemed to be important.



(a) Comparison of performance when selected k values are low

(b) Comparison of performance when selected k values are high

Figure 6.10: Analysis and comparison of execution time when k values are extreme

In this particular experiment, the three comparable approaches were run on the same dataset of mutations and patients of size 500. The aim of this experiment as mentioned was to study the behaviour of the execution times with extremely low and extremely high values of k.

6.6.1 Performance analysis with low values of k

The results of this use-case are depicted in figure 5.9 (a). The results are not very different from the results that the previous use-cases has seen. This is due to the similar range of values selected for k. As in other results, the baseline approach TOP-MATA is observed to show the lowest execution time for all values of k. The trend of execution times is the same in case of OEM, where it shows no significant change in execution times with increasing values of k. Therefore, it can be inferred that for small increases in the values of k and for low values of k, the increase in execution times, for TOP-MATA and the proposed approach is minimal.

6.6.2 Performance analysis with high values of k

This subsection presents an interesting analysis to investigate how scalable the approaches are when the values of k increase. The aim of this analysis is also to answer the research question of how the approaches are impacted by the value of k. The results of this experiment are depicted in figure 5.9 (b). For this study, relatively high values of k , 100,500,1000,1500 were chosen. Evidently, OEM records consistently high execution times, however shows negligible changes as the values of k are doubled. It is essential to note the trend of execution times followed by TOP-MATA and the proposed approach. TOP-MATA shows a steady increase in execution times as the value of k doubles, recording a maximum of 104.23 seconds.

As the value of k increases, the increasing execution times in our TOP-MATA implementation and the proposed approach come from different operations in the respective approaches. In our TOP-MATA, the sequence of operations change once the number of items in maxheap reach k, in that beyond a length of k, the approach uses a single 'if' operation followed by an update of minCos depending upon the top k entries. Therefore, a large part of the execution time is due to the nature of the implementation itself.

In our implementation of TOP-MATA, as the value of k increases the list storing topk pairs is sorted to store the {overlap_coefficient, mutation_1, mutation_2}. This operation is linear in the length of the iterable ie the number of such 3-item sets. Therefore, as the value of k increases, there is a steady increase in execution time. Another common operation that contributes to the execution time is the operation

to retrieve the k largest pairs from the topk heap. The time complexity of this operation is in $O(Nlog(k))$ where N is the length of the iterable. As long as the value of k is small , this operation proves to be inexpensive. But in this use-case, with large values of k, this operation increases the cost and thereby bringing down the efficiency in both TOP-MATA and the proposed approach. However, as a result of the aforementioned reasons, TOP-MATA's performance shows an average level of dependence on the values of k.

Moving on to the performance of the proposed approach, the results of the experiment are illustrated in figure 5.9 (b). The empirical evidences show that as the value of k increases by 5 times, TOP-MATA shows an increase in execution time of approximately 134% whereas in the proposed approach , the execution time increases by approximately 556.25%. This exponential increase in runtime can be attributed to the list append and heapify operations that are performed at every iteration of k.

Once the value of k is reached, the algorithm then performs the operations required to retrieve the updated value of the threshold upper bound for *every* iteration. The time complexity of the above operations contribute to the steep increase in execution time. Therefore, as seen in the results, OEM and TOP-MATA outperform the proposed approach when the k values are very high i.e. equal to greater than n, where n is the number of unique mutations in the input. Another reasoning is the limitations of this domain. It is important to note that , in our specific domain, the requirements are limited to high volume of input data and a k value that is low. However, these time complex operations are required in order to reach the desired level of accuracy. The upcoming section covers a comparison and analysis of performance accuracy in the comparable approaches and why it is favourable in our context to trade a marginal increase in execution time for completeness in results.

6.7 Accuracy evaluation

In order to empirically evaluate the completeness of the proposed approach and the baseline approaches, error rate has been used as the measure. In this problem of top k, it is the recall rate that is more important than precision.

To revisit, recall is defined by the proportion of total number of correct records that are retrieved and precision is given by the proportion of correct records among the retrieved. If we consider the problem of top-k as a binary classification task, it follows that the top k similar pairs belong to the positive class and the rest of the pairs are considered as negative or in other words, 'false positives'. Therefore, our application is an example of a use-case where 100% recall is desired, i.e. all the pairs that have the top k similarity measures have to be retrieved by the algorithm.

In our previous discussions, we highlighted the need to know the correct topk results apriori in-order to build a high recall application. The OEM approach is straightforward and therefore, the results retrieved by OEM serves as a reference to know the correct results of topk. Therefore, to evaluate the completeness, the approaches are examined for the falsely retrieved pairs in the top k results. This implies that the total number of retrieved results will be equal to k for a given execution. To recall, error rate is given by the following:

$$\text{Misclassification rate } = \frac{False\ positives + False\ negatives}{Total}$$

To evaluate the accuracy of results, top 10 similar pairs were retrieved from data inputs of different sizes using both TOP-MATA and the proposed approach. The results were then compared to examine completeness in the result sets.

As seen from the graph in Figure 6.11, the baseline approach TOP-MATA, showed incompleteness in its results for some of the executions. For example, in the execution where the input size is 1000, our TOP-MATA implementation retrieved 2 false positive pairs, implying that 2 pairs of results were discarded as false negatives.

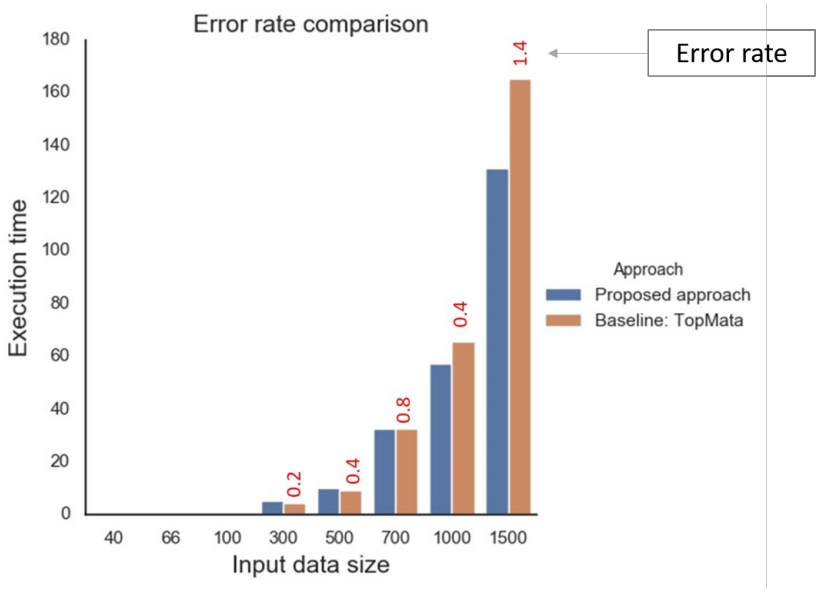


Figure 6.11: Error rate comparison, where error rate is the proportion of false classifications among the total results.

```
Baseline approach : datasize 1000 , top10 results
[[759, 'OTOF_chr2_26754345_T~G', 'OTOF_chr2_26749597_G~A'],
 [681, 'LINC01913_chr2_40917564_A~G', 'LINC01913_chr2_40873538_T~C'],
 [566, 'EXOC4_chr7_133614421_C~A', 'EXOC4_chr7_133347066_A~G'],
 [546, 'PC_chr11_66689240_G~T', 'PC_chr11_66662731_G~A'],
 [495, 'LOC730100_chr2_52635129_G~A', 'MIR4431_chr2_52696451_T~C'],
 [485, 'OTOF_chr2_26754345_T~G', 'TNFRSF9_chr1_7961206_T~C'],
 [479, 'OTOF_chr2_26749597_G~A', 'TNFRSF9_chr1_7961206_T~C'],
 [477, 'OTOF_chr2_26754345_T~G', 'PLXNA4_chr7_132304933_C~G'],
 [476, 'OTOF_chr2_26754345_T~G', 'TRPC4_chr13_38407819_T~C'],
 [475, 'OTOF_chr2_26754345_T~G', 'DCHS2_chr4_155047531_C~T']]
Execution time= 65.52342200279236 seconds
Proposed approach: Datasize 1000, top10 results
[[759, 'OTOF_chr2_26754345_T~G', 'OTOF_chr2_26749597_G~A'],
 [727, 'FTO_chr16_53830465_A~G', 'FTO_chr16_53816275_C~A'],
 [681, 'LINC01913_chr2_40873538_T~C', 'LINC01913_chr2_40917564_A~G'],
 [566, 'EXOC4_chr7_133614421_C~A', 'EXOC4_chr7_133347066_A~G'],
 [546, 'PC_chr11_66662731_G~A', 'PC_chr11_66689240_G~T'],
 [526, 'TNR_chr1_175496350_A~G', 'TNR_chr1_175526374_G~A'],
 [495, 'LOC730100_chr2_52635129_G~A', 'MIR4431_chr2_52696451_T~C'],
 [485, 'OTOF_chr2_26754345_T~G', 'TNFRSF9_chr1_7961206_T~C'],
 [479, 'OTOF_chr2_26749597_G~A', 'TNFRSF9_chr1_7961206_T~C'],
 [477, 'OTOF_chr2_26754345_T~G', 'PLXNA4_chr7_132304933_C~G']]
Execution time= 57.18166422843933 seconds
```

Figure 6.12: Differences in completeness of results between the our baseline TOP-MATA and the proposed approach. As highlighted, the results retrieved by TOP-MATA implementation have exclude some of the top k pairs which are observed to be correctly retrieved by the proposed approach like the pair with overlap coefficients 727 and 526.

Similarly , when top 10 pairs were extracted from data inputs of size 300, 500, 700 and 1,500 TOP-MATA retrieved incomplete results when the proposed approach retrieved accurate results of top 10 pairs.

6.8 Summary of the chapter

This chapter described the experimental configurations and the parameters that were used in the evaluations. Further, we also assessed the performance of the newly devised approach as against the baseline approaches for various selected use-cases and interpreted the empirical evidences with the help of visualizations and tabular data descriptions. The aim of this chapter was to cover the feasibility of the hypothesized solution and to answer the research questions that were raised in the former sections of this study.

Chapter 7

Conclusions and Future work

This study addressed the specific aspect of scalability associated with the task of retrieving the top k pairs of similar entities. In this direction, we proposed a new approach that heuristically reduces the search space complexity thereby optimizing the task of top k retrieval. This newly devised approach is presented in the form of an algorithm which is built by utilizing the monotonicity property of the interestingness measure that is identified as suitable for this problem domain. This research also aimed to examine the impact of input size, k value and cardinality distribution on the execution time. The proposed approach was implemented on a real word dataset in order to establish the applicability of the approach. As part of this scientific work, experimental evaluations were conducted to answer the research questions empirically. Every use-case was further subjected to several parameters and the results were interpreted. The broad implication of the this research shows that the dataset , the underlying domain and the similarity measure of choice collectively influence the design of approach.

More generally, our results appear consistent with the research showing that pruning of the search space optimizes execution time by minimizing the number of false positive comparisons.

The remainder of this chapter includes discussions about the results of the experiments detailed in the previous chapter, followed by a review of the limitations of this study and recommendations for future research.

7.1 Discussions

The experimental evaluation of our proposed approach was aimed at addressing practical aspects of this scientific study like applicability and also generalizability. The interpretations of the results were explained in the previous sections along with a causal analysis was presented. In order to ascertain the generalizability of this work, it is important to understand the implications of these results. As the value of k increases, TOP-MATA as well as the proposed approach showed an increasing trend in the execution time. This suggests that the value of k is directly proportional to execution time in approaches that utilise upperbound based pruning methods.

In OEM however, since there was no dynamic changes to the search space, the execution time showed no correlation with the k value. This suggests that apart from the high time complexity in OEM, a method that is independent of k is beneficial to augmenting the generalizability and domain independence. Even though for smaller input sizes, the proposed approach has an execution time that is approximately 23% more than TOP-MATA, the completeness of the proposed approach makes it a better choice of tool. When the input is larger i.e., for 1000 and 1500, the TOP-MATA implementation shows an average increase in execution time of 18.28% across the 4 chosen values of k. It can be said that a straightforward sequential access of records in the proposed approach and a more aggressive pruning strategy contributes to the lower execution time. Another use-case was the influence of cardinality distribution on execution time in case of TOP-MATA and the proposed approach. The significant difference in runtimes between two equal sized datasets but having different set lengths shows that both the approaches are operationally influenced by this factor. In this particular use-case, TOP-MATA as well as the proposed approach are similarly impacted.

Another use-case that we studied was how the number of iterations taken by different approaches fared. We have shown that, the proposed approach and TOP-MATA

require similar number of iterations for any given value of k. This implies that the difference in execution time then , comes from the computationally intensive operations used in their respective implementations. The results in this use-case also showcase the importance of size-based sorted access in the proposed approach throwing light on the cost of unsorted access. The proposed approach when applied on an unsorted data input requires higher number of iterations to retrieve the top k pairs compared to all approaches and proves to be inefficient. Further, we found that, in the case of larger size dataset and for small values of k , the proposed approach outperforms the OEM and TOP-MATA, in terms of execution time and accuracy.

Based on the quantitative analysis of execution time behaviour, it can be concluded that the size of input, the value of k and the associated cardinality distribution in the dataset are important factors to be considered when selecting an approach to efficiently retrieve the top k similar pair of entities. To answer our research question, the execution time in our proposed approach are directly proportional to these parameters. This experiment adds to a growing corpus of research that highlights the motivation for an efficient top k retrieval tool and the complexity of this problem. This study also reiterates our hypothesis of the significance of a pruning criteria in order optimize the process of top k retrieval.

7.2 Limitations

Even though the search space is iteratively pruned, memory required is still at least equal to the size of the dataset. From the experimental section , we know that there are selected scenarios where our approach works and selected scenarios where our proposed approach does not work. The empirical evidences suggest that , one such use-case where our approach shows sub-optimal performance is when the size of the dataset is small and comparable in value to the value of k. In such a scenario, TOP-MATA is seen to have the lowest execution times and with completeness in results. The proposed approach although comparable execution times, is still higher than TOP-MATA. Apart from this, we also observed that the proposed approach was regrettably, not scalable in k.

The proposed solution only works in the given domain. To elaborate, it has been

developed by keeping in mind a tailored set of requirements. This makes it heavily domain dependent. The data pre-processing, the design of the approach and the suitability are influenced by data of very specific nature. This, although gives accurate results, diminishes the overall generalizability of the approach in itself. Further, the proposed approach has been designed by focusing only on datasets that can be processed in memory. For larger volumes of data, the approach did not terminate execution in a reasonable amount of time. This creates a bottle neck for implementing on largescale datasets.

The execution time in the proposed approach also depends on the distribution of cardinality in the data. As we saw in the experimental evaluations , the dataset with lower overall distribution of cardinality required lesser time to extract top k similar pairs when compared to the dataset with higher overall cardinality.

Another unfortunate inadequacy in our solution is the lack of semantic awareness. This can be better explained with the help of the following example: Consider two items A and B having with cardinalities {486} and {400} respectively, having an overlap co-efficient of value of 86. Let item C have a cardinality of {86} and D have a cardinality of {86}. The overlap co-efficient value between C and D is also 86. The proposed approach assumes quantitative equivalence to the overlap values of the pairs (A,B) and (C,D). But semantically, depending on the dataset domain, in our case, A and B can be regarded as frequently occurring mutations having a rather low value of absolute co-occurrence whereas, the mutations C and D are comparatively less frequent in the dataset and yet shows complete co-occurrence with respect to one another. Such an information may be significant as a finding and unfortunately , our approach does not factor in a weight value to the co-occurrence with respect to the mutual cardinalities of the items in the pair.

7.3 Future work

In this work of research, our hypothesis was that the use of a pruning based, adaptive approach to iteratively reduce the search space , optimizes the top k problem. The empirical evaluations highlighted certain aspects of the study that do not live up to the expectations of reduced execution time. We know that, currently, the execution

time is impacted by the distribution of the associated cardinality. Meaning that the operation of the actual intersection adds significant overhead computationally. One way to empower the approach would be to introduce an intermediate phase for candidate generation and limit the computation of pairwise intersection to only a subset of candidates that are likely to be in the top k set. We recommend that further investigation can be in the direction of using positional filtering techniques used in the context of inverted indices and near duplicate detection to generate a candidate set. We propose that further research could be done to make this parameter have an influence on the proposed approach that is statistically insignificant. Another aspect of this work that could be further researched is ,in making the proposed approach more semantically aware. In light of this, the following ideas could be tested:

1. Instead of assuming quantitative equivalence in pairs of entities that have the same value of overlap, the results can be grouped based on their overlap coefficient. This would also diversify the results and reduce redundancy.

2. In order to better recognise the semantic information in pairs that have equal values of overlap, a weight function can be used to capture the numerical difference in their respective individual frequencies/cardinalities.

Our approach can be used not only in the context of *pairwise* co-occurring mutations but have more widespread applications in identifying more than 2 co-occurring entities. More research on making the approach scalable with big data can be beneficial in furthering its applications in the domain of biomedical data mining. The empirical evidences also show promising results when an adaptive pruning of the search space is performed.

7.4 Summary of the chapter

This chapter was devoted to a high level discussion of the experimental results and their implications as concluding remarks. We also identified and presented the nature and source of the limitations of this study. Finally, we recommended aspects of this research that warrants further investigation.

Bibliography

[1] Griffiths AJF et al. *An introduction to genetic analysis.* 7th edition. 2000.

[2] Ramu Anandakrishnan et al. "Estimating the number of genetic mutations (hits) required for carcinogenesis based on the distribution of somatic mutations". In: *PLoS computational biology* 15.3 (2019), e1006881.

[3] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. "Efficient exact set-similarity joins". In: *Proceedings of the 32nd international conference on Very large data bases*. 2006, pp. 918–929.

[4] Roberto J Bayardo, Yiming Ma, and Ramakrishnan Srikant. "Scaling up all pairs similarity search". In: *Proceedings of the 16th international conference on World Wide Web*. 2007, pp. 131–140.

[5] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. "A primitive operator for similarity joins in data cleaning". In: *22nd International Conference on Data Engineering (ICDE'06)*. IEEE. 2006, pp. 5–5.

[6] William S Cooper. "On selecting a measure of retrieval effectiveness". In: *Journal of the American Society for Information Science* 24.2 (1973), pp. 87–100.

[7] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[8] Qinghua Cui. "A network of cancer genes with co-occurring and anti-co-occurring mutations". In: *PLoS One* 5.10 (2010), e13180.

[9] Dong Deng, Yufei Tao, and Guoliang Li. "Overlap set similarity joins with theoretical guarantees". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 905–920.

[10] Tamer Elsayed, Jimmy Lin, and Douglas W Oard. "Pairwise document similarity in large collections with MapReduce". In: *Proceedings of ACL-08: HLT, Short Papers*. 2008, pp. 265–268.

[11] Brian Eriksson. "Learning to top-k search using pairwise comparisons". In: *Artificial Intelligence and Statistics*. 2013, pp. 265–273.

[12] Ronald Fagin, Amnon Lotem, and Moni Naor. "Optimal aggregation algorithms for middleware". In: *Journal of computer and system sciences* 66.4 (2003), pp. 614–656.

[13] Anna Huang. "Similarity measures for text document clustering". In: *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*. Vol. 4. 2008, pp. 9–56.

[14] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. "A survey of top-k query processing techniques in relational database systems". In: *ACM Computing Surveys (CSUR)* 40.4 (2008), pp. 1–58.

[15] Christopher D Manning, Hinrich Schütze, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge university press, 2008.

[16] Alistair Moffat and Justin Zobel. "Rank-biased precision for measurement of retrieval effectiveness". In: *ACM Transactions on Information Systems (TOIS)* 27.1 (2008), pp. 1–27.

[17] Papias Niyigena et al. "Efficient Pairwise Document Similarity Computation in Big Datasets". In: *International Journal of Database Theory and Application* 8.4 (2015), pp. 59–70.

[18] Mehran Sahami and Timothy D Heilman. "A web-based kernel function for measuring the similarity of short text snippets". In: *Proceedings of the 15th international conference on World Wide Web*. 2006, pp. 377–386.

[19] Hans von Storch and Francis W Zwiers. *Statistical analysis in climate research*.

[20] "What is next generation sequencing". In: *Archives of disease in childhood. Education and practice edition*. 98(6) (2013), pp. 236–238.

[21] Chuan Xiao et al. "Efficient similarity joins for near-duplicate detection". In: *ACM Transactions on Database Systems (TODS)* 36.3 (2011), pp. 1–41.

[22] Chuan Xiao et al. "Top-k set similarity joins". In: *2009 IEEE 25th International Conference on Data Engineering*. IEEE. 2009, pp. 916–927.

[23] Hui Xiong, Mark Brodie, and Sheng Ma. "Top-cop: Mining top-k strongly correlated pairs in large databases". In: *Sixth International Conference on Data Mining (ICDM'06)*. IEEE. 2006, pp. 1162–1166.

[24] Hui Xiong et al. “Exploiting a support-based upper bound of Pearson’s correlation coefficient for efficiently identifying strongly correlated pairs”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 334–343.

[25] Zhong Yang et al. “Adaptive Top-k Overlap Set Similarity Joins”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE. 2020, pp. 1081–1092.

[26] Jian Zhang and Joan Feigenbaum. “Finding highly correlated pairs efficiently with powerful pruning”. In: *Proceedings of the 15th ACM international conference on Information and knowledge management*. 2006, pp. 152–161.

[27] Shiwei Zhu et al. “Scaling up top-k cosine similarity search”. In: *Data & Knowledge Engineering* 70.1 (2011), pp. 60–83.