THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Maximal Independent Sets of a K-mer Space

LERAN MA
SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Mingfu Shao
Professor of Computer Science
Thesis Supervisor

Jesse Barlow
Professor of Computer Science
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

In computational biology, k-mer based algorithms are playing a central role. Various state-of-the-art tools involve k-mer based approaches in solving fundamental problems like sequence alignment, sequence assembly, gene finding, etc. Hence, conducting specific research on k-mers can inspire computational biologists to develop new and innovative ideas. In this thesis, the author aims to understand the structure of the k-mer space by modeling the space as graphs with k-mers as vertices and studying maximal independent sets (MIS) of the graphs. Because a MIS is a sparse and small subset of k-mers selected from the dense and large k-mer space, it can have potential application in hashing and clustering problems. This task is challenging since the size of a k-mer space growth geometrically with respect to k. For 15-mer space, there is more than 1 billion vertices in the graph.

The author proposes three methods for finding maximal independent sets in a k-mer space. The first and most intuitive method involves pairwise comparison between k-mers. The second method implements two heuristics to avoid redundant pairwise comparison: One considers the nearest neighbors of a k-mer, and the other takes advantage of the triangle inequality property of the k-mer space. The last method extends the breadth-first search on a different graph. These three methods are used to find maximal independent sets of small, medium, and large sizes respectively.

Experimental results are reported for k-mer spaces with k ranging from 10 to 15. The most memory- and time-consuming experiment is of a medium-sized maximal independent set of the 15-mer space. The peak memory usage is about 2 GB, and the running time is approximately 15 hours. Source code is available at https://github.com/Shao-Group/kmerspace.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my thesis supervisor Dr. Mingfu Shao from the School of Electrical Engineering and Computer Science at the Pennsylvania State University. It has been a privilege and an honor to work with him. He gave me a lot of valuable suggestions on potential research directions when I faced obstacles designing algorithms. I would also like to thank my honors advisor Dr. Jesse Barlow for his support throughout my career at Schreyer Honors College. Finally, I wish to show my appreciation to my parents for their continuous encouragement through the process of writing this thesis. Thank you.

Introduction

Background Information

As a very important biological genetic material, DNA has a direct impact on the acquired characteristics of organisms. Research on DNA helps people diagnose and predict disease. DNA is made up of nucleotides linked to each other, and there are four types of nucleotides: adenine (A), cytosine (C), guanine (G), and thymine (T). In bioinformatics, researchers mainly study DNA segments obtained through sequencing technologies, from which people can get long or short DNA fragments called reads. By analyzing these reads, researchers can conclude important information about the original DNA. A lot of k-mer-based algorithms have been used when people study reads.

K-mer is a sequence of length k. For instance, the sequence $ACGGA$ is a 5-mer, and $TGCAGG$ is a 6-mer. The k-mer space is a set that collects all k-mers. The monomer space is a set of the 4 basic bases:

$$\{A, C, G, T\},$$

and the 2-mer space is a set of 16 elements:

$$\{AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, TT\}.$$

The purpose of this study is to understand the structure of the k-mer space. Hopefully, this research can facilitate innovations for solving various problems in bioinformatics.

Problem to Solve

Edit distance, or Levenshtein distance, between two strings is the minimum number of insertions, deletions, and substitutions required to transform one string to the other. In this thesis, we model a k-mer space as a graph: each k-mer in the space is a vertex, and two vertices are connected via an edge if their edit distance is shorter than or equal to a parameter d ranging from 2 to $k-1$. We aim to find a maximal independent set (MIS) of this graph. For example,

$$\{AAA, CCC, GGG, TTT\}$$

is a MIS of the 3-mer space with d of 2. These MISs suggest useful information on the structure of a k-mer space and therefore have prospective usage in solving problems like sequence clustering and hashing.

We consider three methods to solve this problem. The first one is a greedy approach that directly applies pairwise comparison on vertices, which is appropriate for finding MISs of small size (with less than 15 vertices). The second method for MISs of medium size (with 15 to 10,000 vertices) uses two heuristics to avoid redundant pairwise comparison: One heuristic takes advantage of the triangle inequality property of the space, and the other refers to information about nearest neighbors of a vertex. The last method involves breath-first search (BFS), which is efficient for MISs of large size (with more than 10,000 vertices).

Experimental results for k-mer spaces with k ranging from 10 to 15 are reported in the Result section.

Methods

Simple Pairwise Comparison Approach

This method stores the MIS in an array, which is initialized empty. A loop is used to iterate over all k-mers in the k-mer space. At each iteration of the loop, a k-mer v is selected from the k-mer space. Then, a nested loop is used to iterate over the MIS array. Each k-mer u from the MIS array is compared with v. If the edit distance between u and v is less than or equal to the parameter d, immediately break the nested loop and continue the outer loop. If after finishing the nested loop, we find that all k-mers in the MIS array have an edit distance larger than d from v, the k-mer v will be added to the MIS array. After the outer loop terminates, the resulting MIS array is a solution to the problem. Figure 1 illustrates the process in pseudocode.

Algorithm 1 Simple Pairwise Comparison

```
initialize an empty MIS array
for each k-mer v in the k-mer space do
    isMapped = false
    for each k-mer u ∈ MIS do
        if edit_distance(u, v) ≤ d then
            isMapped = true
            break
        end if
    end for
    if isMapped == false then
        add v into MIS
    end if
end for
output MIS
```

Figure 1. pseudocode for the simple pairwise comparison approach

We now show that this algorithm is correct, i.e., it gets an MIS. K-mers in the MIS have an edit distance greater than d from each other. Accordingly, in the graph model of the k-mer space, they are not connected directly with each other. For each k-mer v not included in the MIS, we can find at least one k-mer u in the MIS has an edit distance less than or equal to d from v, which implies that v is directly connected to at least one k-mer in the MIS of the graph.

The space complexity of the algorithm is $O(|MIS|)$ where $|MIS|$ denotes the size of the MIS. Because we conduct pairwise comparison between all k-mers in the k-mer space and the k-mers in the MIS, the running time of the algorithm is $O(|V| \cdot |MIS|)$ where $|V|$ is number of vertices in the graph.

Improved Algorithm to Avoid Redundant Pairwise Comparison

From the running time analysis of the previous approach, we know that as the size of the MIS increases, the running time grows. Because $|V|$ grows exponentially with respect to k, such increase of the MIS size can significantly impair the performance of the algorithm for spaces with a large k value. Therefore, we consider the following two heuristics as an extension to the first method to avoid unnecessary pairwise comparison.

The first heuristic uses the property that a k-mer probably have a shared "mapping" with its nearest neighbors. Here, we say a k-mer v is mapped to a k-mer u in the MIS if vu is an edge in the graph. A k-mer can be mapped to multiple k-mers, but we only keep record of one valid mapping to save both the time and the space. Nearest neighbors of a k-mer v refer to a small subset of v's neighbors which contains only k-mers with edit distance 1 from v. For example, the set of nearest neighbors of the 3-mer AAA is

$$\{AAC, AAG, AAT, ACA, AGA, ATA, CAA, GAA, TAA\}.$$

If the 3-mer AAA is selected into the MIS, all the above 9 k-mers will have a shared mapping AAA. This property suggests a modification to the previous method: using an additional array to store the mapping information of all k-mers in the k-mer space. After a k-mer v is iteratively selected from the k-mer space, we immediately compute the nearest neighbors of v, find their mappings in the mapping array, and check if v share a mapping with them. If so, we therefore directly conclude that v should not be included in the MIS and then store the found mapping of v into the array. Otherwise, we continue with the second heuristic.

Before introducing the second heuristic, we need to prove that the k-mer space has the triangle inequality property for edit distance. Imaging the k-mer space as a graph where each vertex is a k-mer, we let $d_{u,v}$ denote the edit distance between vertex u and vertex v. Trivially, $d_{u,v}$ is equal to $d_{v,u}$. For any three vertices a, b, and c, the two following inequality conditions hold true.

$$d_{a,b} \leq d_{a,c} + d_{c,b} \quad ①$$

$$d_{a,b} \geq |d_{a,c} - d_{c,b}| \quad ②$$

Because $d_{a,b}$ is the minimum number of edits used to convert a to b, any conversion from a to b must have a cost of at least $d_{a,b}$ edits. Therefore, the total number of edits, $d_{a,c} + d_{c,b}$, taken to first convert a to c and then convert c to b must be greater than or equal to $d_{a,b}$. The inequality relation ① is true. Similarly, we can deduce the below two inequalities.

$$d_{a,b} + d_{b,c} \geq d_{a,c}$$

$$d_{b,a} + d_{a,c} \geq d_{b,c}$$

Thus, we know

$$d_{a,b} \geq d_{a,c} - d_{b,c} \text{ and } d_{b,a} \geq d_{b,c} - d_{a,c}.$$

The inequality condition ② is also true.

Based on this triangle inequality property, the second heuristic is established as follows. We let σ_k denote the k-mer that is purely composed of the base σ. For instance, a_3 is the 3-mer AAA, and g_5 is the 5-mer $GGGGG$. Then, we know the following relations hold true for any two k-mers u and v and for any $\sigma \in \{a, c, g, t\}$.

$$d_{u,v} \geq d_{u,\sigma_k} + d_{\sigma_k,v}$$

$$d_{u,v} \leq \left|d_{u,\sigma_k} - d_{\sigma_k,v}\right|$$

These two inequalities suggest the upper and lower bounds for the value $d_{u,v}$.

$$\max_{\sigma}\left(\left|d_{u,\sigma_k} - d_{\sigma_k,v}\right|\right) \leq d_{u,v} \leq \min_{\sigma}\left(d_{u,\sigma_k} + d_{\sigma_k,v}\right) \qquad ③$$

The calculation of d_{u,σ_k} is simple. As we defined before, σ_k is the k-mer purely composed of the base σ. Hence, the minimum modification needed to transform u to σ_k is substituting all characters in u that is not σ with σ. We know that

$$d_{u,\sigma_k} = k - \text{number of } \sigma \text{ contained in } u.$$

Therefore, when we put a k-mer u into the MIS, we also store the following four values: d_{u,a_k}, d_{u,c_k}, d_{u,g_k}, and d_{u,t_k}. Every time we need to find a mapping of a vertex v, we first calculate d_{v,a_k}, d_{v,c_k}, d_{v,g_k}, and d_{v,t_k}. Then, we loop through the MIS. For each k-mer u in the MIS, we check if the parameter d is within the range of $d_{u,v}$ specified in ③. If so, we calculate the exact edit distance $d_{u,v}$ and compare it with d. Otherwise, if d is less than $\max_{\sigma}\left(\left|d_{u,\sigma_k} - d_{\sigma_k,v}\right|\right)$, we directly conclude that d is less than $d_{u,v}$ and u is a mapping of v. If d is greater than $\min_{\sigma}\left(d_{u,\sigma_k} + d_{\sigma_k,v}\right)$, we then know that d is greater than $d_{u,v}$ and u is not a valid mapping of v.

Algorithm 2 Heuristic Pairwise Comparison

initialize empty MIS and $mapping$ arrays
initialize empty $d\sigma$ arrays $\forall\sigma \in \{a, c, g, t\}$
for each k-mer v in the k-mer space **do**
  $isMapped = false$
  for each nearest neighbor u of v **do**
    if $edit\_distance(mapping[u], v) \leq d$ **then**
      $isMapped = true$
      $mapping[v] = mapping[u]$
      break
    end if
  end for
  if $isMapped == true$ **then**
    continue
  end if
  $v\_d\sigma = edit\_distance(v, \sigma_k)\ \forall\sigma \in \{a, c, g, t\}$
  for each k-mer $u \in MIS$ **do**
    if $\exists\sigma(abs(v\_d\sigma - d\sigma[u]) > d)$ **then**
      continue
    end if
    if $\exists\sigma(v\_d\sigma + d\sigma[u] \leq d)$ or $edit\_distance(u, v) \leq d$ **then**
      $mapping[v] = u$
      $isMapped = true$
      break
    end if
  end for
  if $isMapped == false$ **then**
    add v into MIS
    $mapping[v] = v$
    $d\sigma[v] = v\_d\sigma\ \forall\sigma \in \{a, c, g, t\}$
  end if
end for
output MIS

Figure 2. pseudocode for the heuristic pairwise comparison approach

The second heuristic takes advantage of the triangle inequality property. The program only does pairwise edit distance comparison when the parameter d fall into a proper range of values. Figure 2 shows the complete pseudocode for the second approach.

BFS Approach

The third approach implements BFS. We think of another graph with both k-mers and (k-1)-mers as vertices. Two vertices are connected to each other if their edit distance is 1, except that there is no edge between (k-1)-mers. For example, if we take $k = 4$, the graph will contain both 4-mers and 3-mers. There is no edge between any pair of 3-mers. Every pair of 4-mers are connected if one 4-mer can be obtained from the other 4-mer by substituting one of the bases. Every pair of 4-mer and 3-mer is connected if the 4-mer can be obtained by inserting 1 base to the 3-mer.

The goal is to find the objective MIS by exploring this new graph. During the exploration, a $distance$ array is used to store the length of the shortest path from a vertex to any vertex in the MIS. Once initialized, elements in this array can only decrease or remain unchanged.

Starting by initializing all fields in the $distance$ array to infinity, we repeat the following three steps until all k-mers in the graph is explored.

1) Pick an unexplored k-mer u from the graph.
2) Add u to the MIS.
3) Explore the neighborhood of u with a depth of d and update the $distance$ array.

The essence of this approach is representing the edit distance between two k-mers with the length of the shortest path between them. In the new graph with both k-mers and (k-1)-mers, one substitution is represented by an edge between two k-mers. Deletion and insertion are represented by the edge between a k-mer and a (k-1)-mer. We make a deletion by walking from a

k-mer to a connected (k-1)-mer, and we perform an insertion by going from a (k-1)-mer to a connected k-mer.

When calculating the edit distance, we consider the three operations: substitution, deletion, and insertion. We are only interested in the minimum number of edits needed to transform one k-mer to another, despite the order of the operations. Because the length of two k-mers are the same, the number of insertions is always equal the number of deletions when we convert one k-mer to another. Although we cannot perform more than one deletion or insertion in a row using only k-mers and (k-1)-mers, we can always change the order of operations to alternate between an insertion and a deletion. For example, the transformation from the 5-mer $TGATT$ to the 5-mer $ATTGA$ can be represented by the following shortest path of length 4.

$$TGATT \rightarrow GATT \rightarrow GATTG \rightarrow ATTG \rightarrow ATTGA$$

Instead of consecutively performing two deletions at the beginning and two insertions at the end, we alternate between one deletion and one insertion.

This BFS approach is correct. Every time an unexplored k-mer is selected into the MIS, we explore its neighborhood with a depth d. Therefore, all k-mers remaining unexplored must have an edit distance more than d from all vertices in the MIS, and all explored k-mers must have an edit distance less than or equal to d from a vertex in the MIS. Hence, k-mers in the MIS are at least $d + 1$ edits apart. We ensure that in the original k-mer space graph model, there is no edge between any two k-mers in the resulting MIS, and all k-mers outside the MIS are connected to at least one k-mer in the MIS.

The space complexity of this algorithm is $O(|V|)$, where $|V|$ is the total number of k-mers and (k-1)-mers in the graph. Because for large k values, the graph is too dense and too large to be stored in memory, we choose to trade increased running time with decreased space usage and

compute edges on the fly instead of storing them. The space is mainly used to store the distance array, which grows linearly with respect to the number of vertices. Besides, according to BFS, each vertex can be explored at most d times. Every time we explore a vertex, we compute its neighbors, and the number of neighbors of a vertex increases linearly with respect to k. Thus, the running time for this approach is $O(dk|V|)$. (See Figure 3 for pseudocode.)

Algorithm 3 BFS Variant

```
initialize an empty distance array
distance[v] = ∞ for all k-mers v
distance[u] = ∞ for all (k-1)-mers u
for each k-mer v in the k-mer space do
    if distance[v] == ∞ then
        output v as a vertex in the MIS
        distance[v] == 0
        initialize an empty frontier queue
        frontier.add(v)
        while frontier is not empty do
            u = frontier.pop()
            for each neighbor w of u do
                if distance[w] > distance[u] + 1 then
                    distance[w] = distance[u] + 1
                    if distance[u] + 1 < d then
                        frontier.add(w)
                    end if
                end if
            end for
        end while
    end if
end for
```

Figure 3. pseudocode for the BFS approach

Comparison between the Three Approaches

The three approaches' running time varies with respect to the parameter d. d specifies the size of the corresponding MIS. If we take relatively large d values, independent k-mers are required to be further apart. As a result, we will have a sparse MIS with small size. Conversely, small d values give us dense MISs with large sizes.

The first approach, Simple Pairwise Comparison, uses the least memory, $O(|MIS|)$, and has a running time of $O(|V| \cdot |MIS|)$. Thus, this approach is both time-efficient with relatively large d values, which yield sparse MIS of size less than 15. However, with medium or small d values, this approach performs too many redundant expensive pairwise comparison due to the large size of the MIS.

Hence, in the second approach, we introduce two heuristics as an extension of the first method to handle medium d values, which corresponds to MISs with approximately 15 to 10,000 vertices. Although the two heuristics help avoid unnecessary edit distance calculations, they also introduce some overhead like computing the nearest neighbors and the edit distance to the k-mer σ_k. Thus, if the MIS includes less than 15 k-mers, the first approach performs better than the second one.

For small d values which result in MISs with more than 10,000 vertices, the first two approaches are slower than the third BFS approach, which has a running time $O(dk|V|)$. This approach does not include the expensive process of edit distance calculation. Instead, the edit distance between two k-mers is represented as the length of the shortest path in a graph. The parameter d indicates the maximum number of explorations for each vertex. Hence, the running

time of the BFS method decreases as d decreases, whereas the first two approaches' running time increases as d decreases.

Results

Experiments are conducted for k ranging from 10 to 15. Table 1 reports the size of the resulting MIS.

Table 1. MIS size for $k \in \{10, 11, 12, 13, 14, 15\}$

d \ k	10	11	12	13	14	15
1	262144	1048576	4194304	16777216	67108864	268435456
2	11743	40604	141943	500882	1782677	6388106
3	1463	4574	14522	46908	153767	510118
4	242	668	1894	5517	16440	49992
5	57	133	338	879	2346	6486
6	17	38	79	188	448	1107
7	9	13	28	54	112	251
8	4	4	12	20	37	75
9	4	4	4	11	14	30
10		4	4	4	10	13
11			4	4	4	8
12				4	4	4
13					4	4
14						4

For each row with a same d value, approximately the size of the MIS increases geometrically with respect to k, which is consistent with the geometric growth in the size of the k-mer space with a constant ratio of 4. Figure 4 depicts the growth of the MIS with respect to k for $d = 2$.

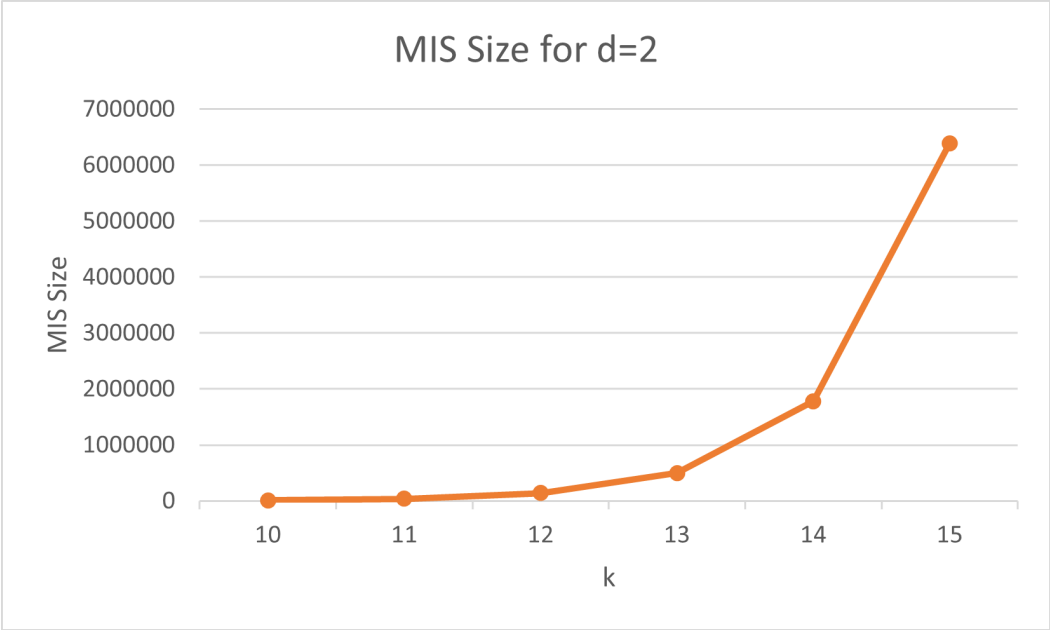

Figure 4. MIS size for $d = 2$

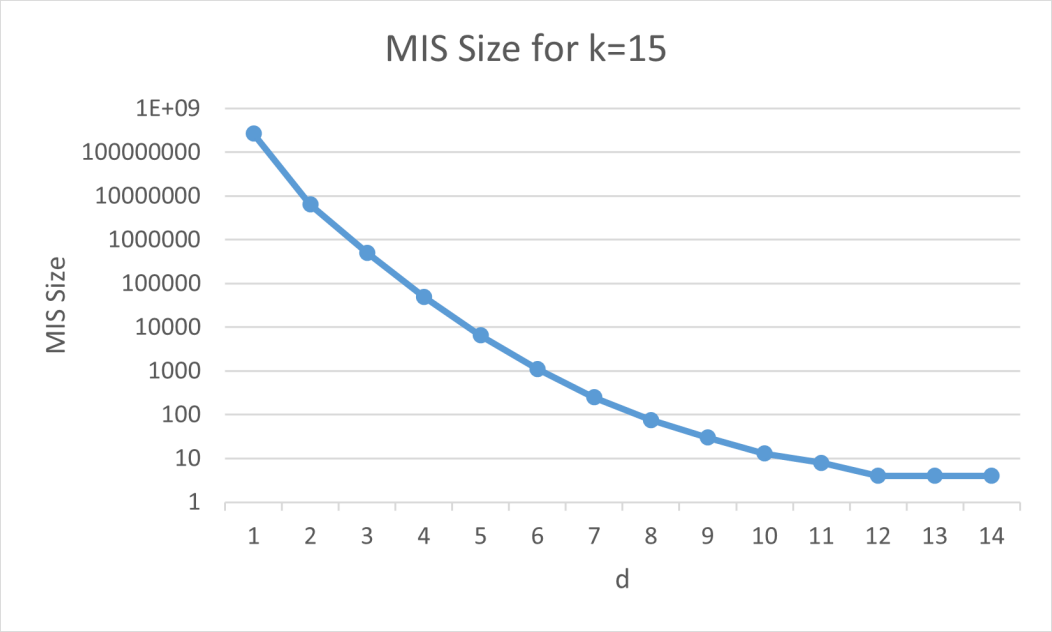

Figure 5. MIS size for $k = 15$

For each column with a same k value, the MIS size decreases geometrically with respect to the growth of d. As we discussed before, d specifies the sparsity of the resulting MIS. With larger d value, fewer vertices are selected into the MIS. Figure 5 depicts the decrease of MIS size with respect to d for $k = 15$.

Regarding the performance of the algorithm, the primary concern is the running time. Table 2 reports the fastest approach to calculate the MIS for each combination of k and d, and the following Table 3 and Table 4 is the corresponding time and memory usage.

Table 2. Fastest Approach for each Combination of k and d

d \ k	10	11	12	13	14	15
1	3	3	3	3	3	3
2	3	3	3	3	3	3
3	3	3	3	3	3	3
4	2	2	2	3	3	3
5	2	2	2	2	2	2
6	1	2	2	2	2	2
7	1	1	2	2	2	2
8	1	1	1	1	2	2
9	1	1	1	1	2	2
10		1	1	1	1	2
11			1	1	1	1
12				1	1	1
13					1	1
14						1

1, 2, and 3 indicates the number of the approach: 1 for Simple Pairwise Comparison, 2 for Heuristic Pairwise Comparison, and 3 for BFS.

Table 2 together with Table 1 suggests a general strategy to choose appropriate approach for specific combination of k and d. The Simple Pairwise Comparison approach is efficient for d values within the interval $[k-4, k-1]$ since these d values result in a MIS with less than 15

vertices approximately. The BFS method is best for d values between 1 and 4 because such small d leads to a too large MIS to perform pairwise comparison. Any b value approximately between 4 and $k - 4$ is preferred to be handled by the Heuristic approach since the BFS approach's running time grow linearly with respect to d.

Table 3. Running Time (Seconds)

d \ k	10	11	12	13	14	15
1	5	25	114	528	3382	14028
2	11	46	202	977	5444	22380
3	18	78	344	2161	10317	46701
4	8	55	460	4191	21801	36614
5	4	25	152	1409	11887	53478
6	3	17	94	659	4423	23922
7	1	13	77	504	3008	14864
8	1	7	52	438	2422	11308
9	0	5	31	258	2128	10003
10		4	23	210	1809	9197
11			21	187	1225	7322
12				184	988	5548
13					931	4743
14						4332

Paying attention to the rows of Table 3, we can see that the running time generally increases geometrically with respect to k. The current computation bottleneck is the case for $k = 15$ and $d = 5$, which takes 53478 seconds, approximately 15 hours using the Heuristic Pairwise Comparison approach.

Table 4. Memory Usage (kB)

d \ k	10	11	12	13	14	15
1	3648	4568	8524	23916	85316	331164
2	3544	4596	8364	24012	85396	331220
3	4008	5148	8888	24336	85984	331908
4	5460	11508	36224	29512	92524	341752
5	5352	11468	36016	134416	527728	2100852
6	1952	11584	36272	134484	527672	2100688
7	1880	2012	36164	134480	527688	2100568
8	1920	1868	1952	1920	527772	2100568
9	1980	2012	3464	1920	527760	2100548
10		1980	1868	3456	1964	2100636
11			2032	3396	3184	3308
12				1976	3380	1992
13					3404	1964
14						3384

According to Table 4 and Table 2, the Heuristic approach always has the largest the memory overhead compared with other approaches. The extra memory is mainly used to save the mappings of all k-mers. The peak memory usage is about 2,100,000 kB, approximately 2 GB for some cases of $k = 15$.

Discussion

We have proposed a problem of extracting MIS as a representative substructure from a graph model of a k-mer space. Three approaches are designed to efficiently solve the problem for different ranges of parameter d. The first one is a simple greedy algorithm using pairwise comparison. The second approach extends the first by implementing two heuristics to avoid some redundant comparison. The third approach views the edit distance as the shortest path in a different graph and applies BFS. Experiments are done for k ranging from 10 to 15. The computation bottleneck occurs at $d = 5$ for the 15-mer space using the second approach. The corresponding peak running time is approximately 15 hours, and the peak memory usage is about 2 GB.

Due to the limitation of the current approaches, the largest space our approaches can handle with reasonable costs of time and space is the 15-mer space. For future work, we would like to extend this study to larger spaces like 20-mer space. Considering the current computation bottleneck, one potential improvement is to design more heuristics to partition the k-mer space so that only certain small subset of k-mers will be involved in the expensive pairwise comparison process.

Another idea is to revise the BFS approach. Because there are only 4 bases, the new graph built for the BFS is highly symmetric considering the permutation of bases. For example, the search trees for the 5-mers $AGAAC$ and $TATTG$ are essentially the same, expect that the bases A, G, and C is replaced with T, A, and G respectively. In our BFS approach, lots of explorations of symmetric vertices are actually done redundantly if we can take advantage of symmetry. (1)

could provide some insights on how to find all symmetric relationships in the k-mer space using deterministic finite automata.

Regarding the application of our MISs, one potentiality is their usage in read clustering problem. The choice of centers is essential in clustering problems. (2) and (3) are two applications of k-mer based clustering algorithms. Because a MIS contains a group of representatives for a k-mer space with a certain edit distance apart, we can the initialize centers using the MIS. Besides, the MIS may also be used to generate keys for read hashing problems, facilitating fast query of reads. (4) and (5) are two articles related to k-mer based hashing algorithm.

Conclusion

We have presented an idea of extracting MIS as a representative substructure from a graph model of a k-mer space. Three different approaches are designed to efficiently find MISs with different ranges of parameter d. The first approach is a simple greedy algorithm using pairwise comparison. The second approach improves upon the first by implementing heuristics based on two properties of k-mer space: shared mapping of nearest neighbors and triangle inequality. The third approach applies BFS. Experiments are done for k ranging from 10 to 15. The computation bottleneck occurs at $k = 15$ and $d = 5$ using the second approach. The corresponding peak running time is approximately 15 hours, and the peak memory usage is about 2 GB. Furthermore, the resulting MIS can be potentially useful for solving read clustering and hashing problems.

Bibliography

1. Schulz, K., Mihov, S. Fast string correction with Levenshtein automata. *IJDAR* **5**, 67–85 (2002). https://doi.org/10.1007/s10032-002-0082-8
2. Kim, C.S., Winn, M.D., Sachdeva, V. *et al.* K-mer clustering algorithm using a MapReduce framework: application to the parallelization of the Inchworm module of Trinity. *BMC Bioinformatics* **18,** 467 (2017). https://doi.org/10.1186/s12859-017-1881-8
3. Kirk, J.M., Kim, S.O., Inoue, K. *et al.* Functional classification of long non-coding RNAs by *k*-mer content. *Nat Genet* **50,** 1474–1482 (2018). https://doi.org/10.1038/s41588-018-0207-8
4. Hamid Mohamadi, Justin Chu, Benjamin P. Vandervalk, Inanc Birol, ntHash: recursive nucleotide hashing, *Bioinformatics*, Volume 32, Issue 22, 15 November 2016, Pages 3492–3494, https://doi.org/10.1093/bioinformatics/btw397
5. Enrico Petrucci, Laurent Noé, Cinzia Pizzi, and Matteo Comin. Iterative Spaced Seed Hashing: Closing the Gap Between Spaced Seed Hashing and k-mer Hashing. *Journal of Computational Biology*. Feb 2020. 223-233. http://doi.org/10.1089/cmb.2019.0298

ACADEMIC VITA

Leran Ma

EDUCATION

Bachelor of Science in Computer Science **Expected 5/22**

The Pennsylvania State University, University Park, PA

- Schreyer Honors College (Spring 22, Fall 21, Spring 21, Fall 20)
- Behrend Honors Program (Spring 20, Fall 19, Spring 19, Fall 18)

EXPERIENCE

Undergraduate Research Assistant in Research Lab **6/21–present**

The Pennsylvania State University, State College, PA

- Collaborating with doctoral students and postdoctoral scholars on research projects under the supervision of Prof. Mingfu Shao
- Writing formal research manuscripts for academic conferences RECOMB and ISMB
- Learning the application of various algorithms in Bioinformatics

Hackthon: HackPSU **11/19**

The Pennsylvania State University, State College, PA

- Learned the application of online APIs
- Acted as a group leader when completing a programming project

IT Internship **6/19–7/19**

Shineway Technology, Beijing, China

- Analyzed experimental data and simulated corresponding curves using MATLAB at the Research and Development department
- Recorded the order information and drew up contracts at the Business Department

Bioinformatic Internship **6/17–7/17**

Chinese Academy of Science, Beijing, China

- Performed accurate biology experiments to decompose given unknown protein to peptides
- Examined experimental data and performed computation to find the specific amino acid sequence of the unknown protein using MATLAB
- Wrote formal experiment reports

SKILLS

Java (advanced)	C++ (advanced)	C (advanced)
Python (advanced)	SQL (advanced)	PHP (beginner)
MATLAB (beginner)	R (beginner)	VHDL (intermediate)
Verilog (intermediate)	Visual Basic (intermediate)	

COMMUNITY SERVICE

Volunteer **7/17, 7/18, 7/19**

Cuncaochunhui Old-Age Home, Beijing, China

- Chatted with elders
- Prepared meals for elders
- Regular cleaning

ACADEMIC PROJECTS

Maximal Independent Set in K-mer Space **6/21–present**

- Modelling the set of all nucleic acid sequences of given lengths as graphs
- Applying various algorithms to find the maximal independent set for graphs with trillions of nodes
- Learning optimization techniques to save time and space usage of a program
- Finding various kinds of centers for clustering problems and keys for hashing problems of nucleic acid sequences

TCP Variants **4/21**

- Created a computer network topology using C++ and NS-3 APIs
- Simulated TCP and UDP network traffic
- Traced the congestion window size for TCP variants Tahoe, Reno, and New Reno

Dynamic Storage Allocator **3/21**

- Applied segregate fit and slab allocation to decrease fragmentation in virtual memory

ScatterGather Driver **12/20**

- Created a device driver sitting between a virtual application and virtualized hardware devices using C
- Simulated online storage systems like OneDrive or Box

European Soccer Database **12/20**

- Created an online interactive database using PHP and MySQL

Pipelined CPU **12/20**

- Created a five-stage pipelined CPU using Verilog
- Performed hardware simulation using Xilinx design package for FPGAs

Count Vowel Project **12/19**

- Created a web-based application using Spring Framework

Drivability Project **11/19**

- Designed a Java application to estimate the road condition
- Connected online APIs with Java program using Maven

Library Book Management Project **10/19**

- Connected given databases with Java program using Maven

- Manipulated given databases using SQL

Connect 4 **9/19**

- Designed the two-player computer game Connect 4 using JavaFX

OTHER SIGNIFICANT COURSES

Public speaking	Technical writing	Calculus
Discrete math	Statistics	Linear algebra