CS299 INNOVATION LAB

Abhinav Dutta 1901CS02

Guide: Prof. Raju Halder

ANALYSIS OF PROGRAMS/TEXT

Introduction

- In this project we will build methods to process the textual content of programs/documents to find some desirable properties. Our initial target is to detect similarity between two documents. Existing software does it by masking all variables with a special character. We improve upon this by allowing the user to define how they want to mask the variables. We , then realized that by defining a relationship between the various masking categories, other security - related issues can also be solved.

Motivation

Current plagiarism detection tools replace all variables by a special character. This is not very flexible since the user might want to treat different variable differently based data type or his own preference.

Current plagiarism detection tools don't allow for cost-accuracy trade-offs. So it becomes infeasible to work with large documents or large collection of documents. Although it is straightforward to allow the users to select the window size while working with large files.

Related Work

Winnowing: Local Algorithms for Document Fingerprinting (Saul Schleimer et.al 2003) Introduces the idea of minimizers for document fingerprinting which is used in MOSS

Weighted minimizer sampling improves long read mapping (Chirag Jain et.al 2018) Introduces the idea of frequency weighted minimizers which is used to build the tool Winnowmap (used for gene sequence analysis).

Contributions

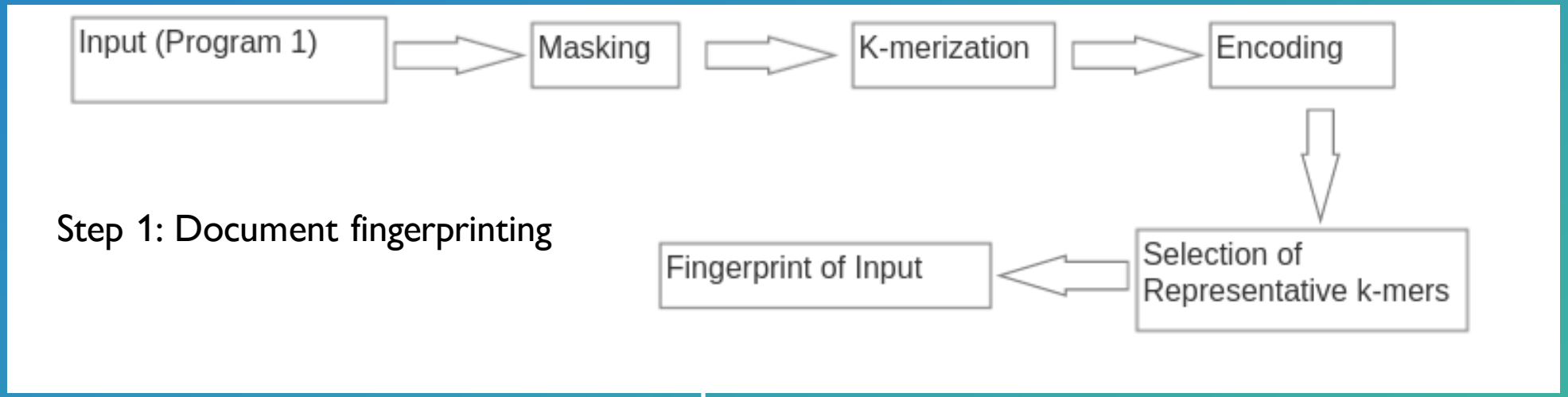
Implemented Colour encoded matchings

Devised a novel masking scheme which can be used to detect vulnerabilities that are not otherwise detected by compilers (for eg. if public variables can store the value of private variable it is not detected but in reality this is a vulnerability). Also such masking schemes improves plagiarism detection.

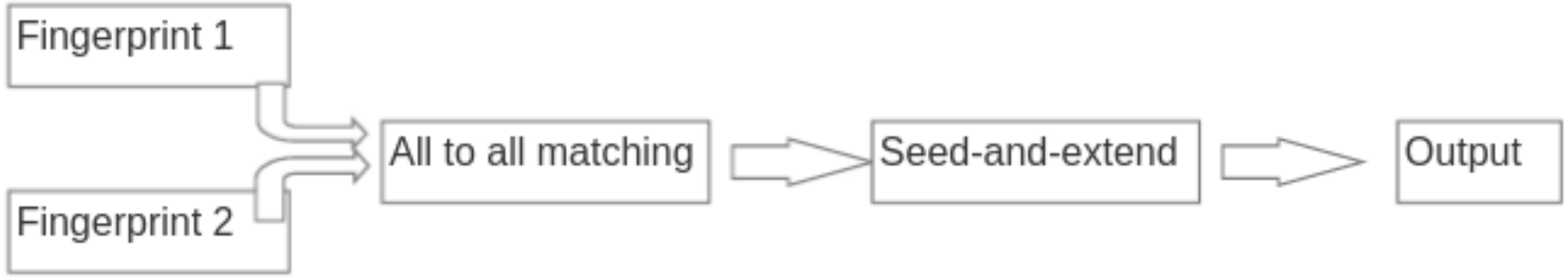
Implemented frequency weighted minimizersrs for plagiarism tool which has not been done used in MOSS and other plagiarism tools based on k-mers (preferential selection of rare k-mers)

The winnowing paper and also github implementations of plagiarism tools based on minimizers has complexity O(n*w) (where n is number of characters and w is window size) . However this can be improved to O(n) as suggested in the Winnowmap paper (2018) using monotone queues and this is what I have used in this project.

WORKFLOW



Step 1: Document fingerprinting



Step 2 : Comparision

Overview of Similarity detection

- We tokenize the input (text/masked code) into k-mers. And we select some of the k-mers as representative of the sequence. This step reuires 2 parameters k and w. k controls the selectivity-specificity trade off whereas w primarily controls the cost-accuracy trade off. (Out of w adjacent k-mers atleast 1 k-mer is always selected). There are 2 schemes used to select the representative k-mers :
- Minimizers (S.Schleimer, 2003) - Current similarity detectors (MOSS) like use this.
- Syncmers (R. Edgar , 2021) - This is another scheme which is selects more evenly spaced k-mers as representatives. Has not yet been used in plagiarism tools.

In my project I have mainly focused on minimizers but can be easily extended to syncmers

- Step 1: k-mers are selected according to the schemes (Minimizer/frequency weighted minimizer).
- Step 2: The selected k--mers from two files are compared and the similarity metrics (Jaccard index) are determined and the similar text is highlighted.

Algorithms

- **Masking algorithm**

The user associates a category with every variable. The user can define a poset on the categories. In this case we have defined the relation based on sensitivity of data.

If var1= f(var2,var3,...vark) then category(var1)=lowest upper bound(category(var1,var2,...vark)).

For. eg if the value stored by private var1 (say password) is more sensitive than the value stored by public var2(say name) than category of var1>= category of var2. Relations maybe be assigned based on data type as well like integers and floats maybe masked separately such that category(float)>category(integer) in the poset (since a float can store a int but not vice versa)

In case the poset does not form a lattice, instead of lowest upper bound ,I have considered any upper bound randomly. We formulate the problem of finding Join(a,b) following as a graph based question : Finding the lowest common ancestor of a and b in the poset graph. (Note : Normal lowest common ancestor algorithm does not work here since the graph is not a tree – each vertex has multiple parents)

Pseudo Code :

Input - the poset as a graph with directed edges stored in adjacency list adj[], source vertex

FindJoin(vertex a,vertex b)

```
Mark all the ancestors of a with blue
Mark all the ancestors of b with black
DFS to find the height of all vertices
V = Choose the vertex of greatest height that is both blue and black
Return V
```

Algorithm 1: Standard procedure for computing minimizers

Input: w, $arr[\,]$ //array of k-mers in the order as they occur in seq.
Output: $M[\,]$ //array of minimizers

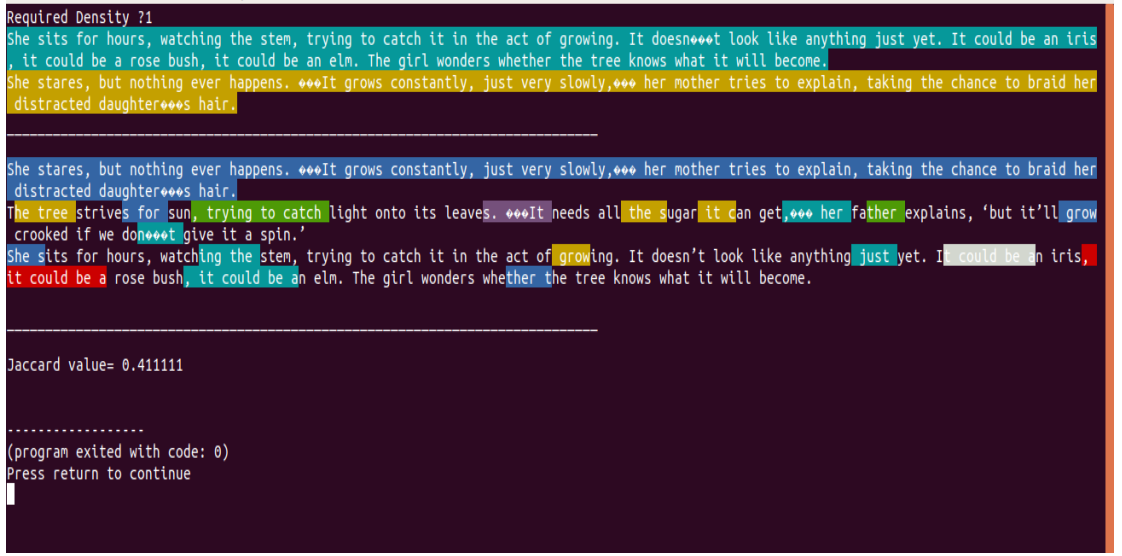
```
1  Array M ← [ ]; Deque Q ← [ ];
2  for i ← 1 to arr.size()
3      //each k-mer is represented by a pair of its position and order
4      ⟨pos, ord⟩ ← ⟨ i, getOrder(arr[i]) ⟩;
5      while ¬Q.empty() and Q.back.second > ord
6          Q.pop_back();
7      Q.push_back( ⟨pos, ord⟩ );
8      if Q.front.first ≤ i − w then
9          while Q.front.first ≤ i − w
10             Q.pop_front();   // discard out-of-range k-mer
11         furtherPop(Q);
12     sample (Q.front, M);
13     furtherSample (Q, M);   // process ties
14 Function sample (min, M)
15     if M.back ≠ min then
16         M.push_back (min);   // add to index
17 Function getOrder (mer)
18     return h(mer);   // hash fn
19 Function furtherPop (Q)
20     return;   // do nothing
21 Function furtherSample (Q, M)
22     while Q.front.second = Q.next_to_front.second
23         Q.pop_front();
24         sample (Q.front, M);
```

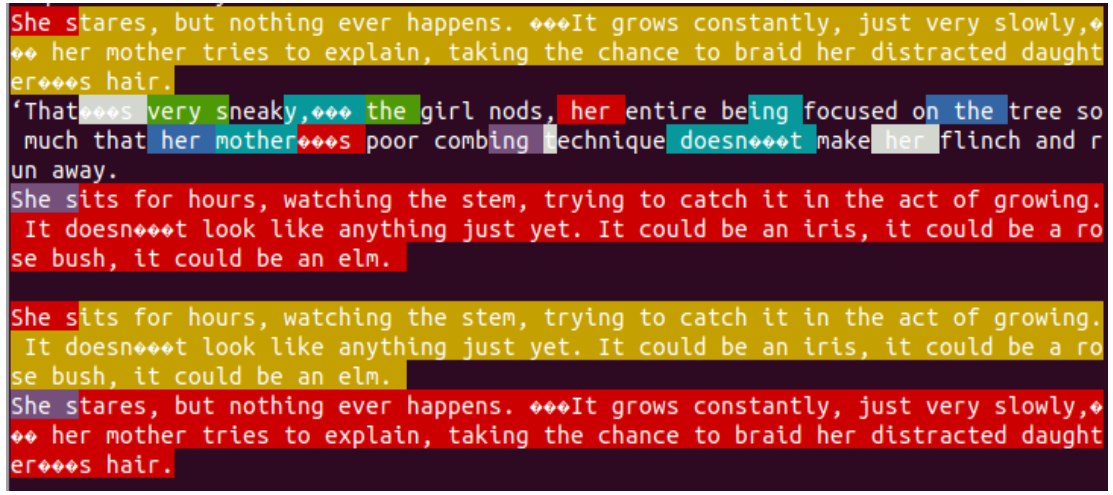
Seed-and-Extend Pseudo Code:
Input - location of common k-mers in file 1 and 2
Output - pair<int,int> match1, match2 -which contains the regions which have matched
Seed-and-Extend(pos1,pos2)

```
i=pos1, j=pos2
while(file[i]==file[j])
        i--;  j--;
left1=i; left2=j;

i=pos1,j=pos2;
while(file[i]==file[j])
        I++; j++;
right1=i; right2=j;

Return <left1,right1> , <left2,right2>
```

Credit : Chirag Jain
Winnowmap paper

Experiments (similarity detection)

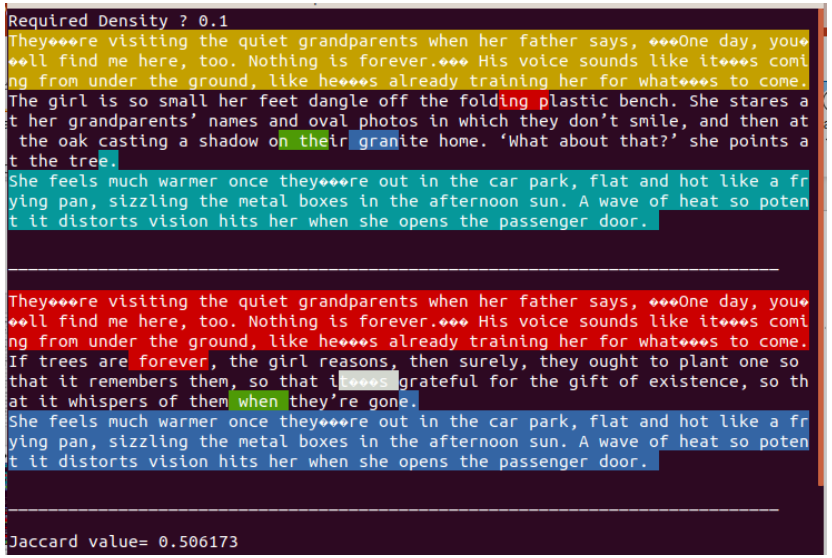


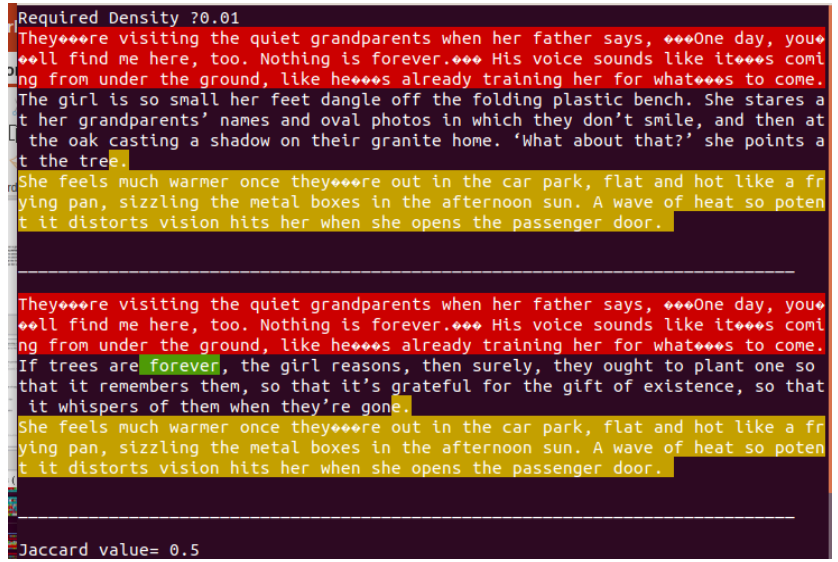


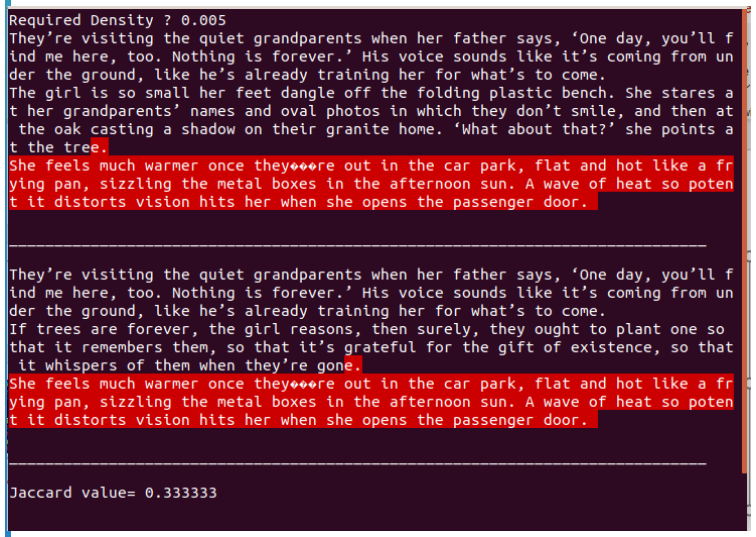
Small k-values (k=4) has many small random matches

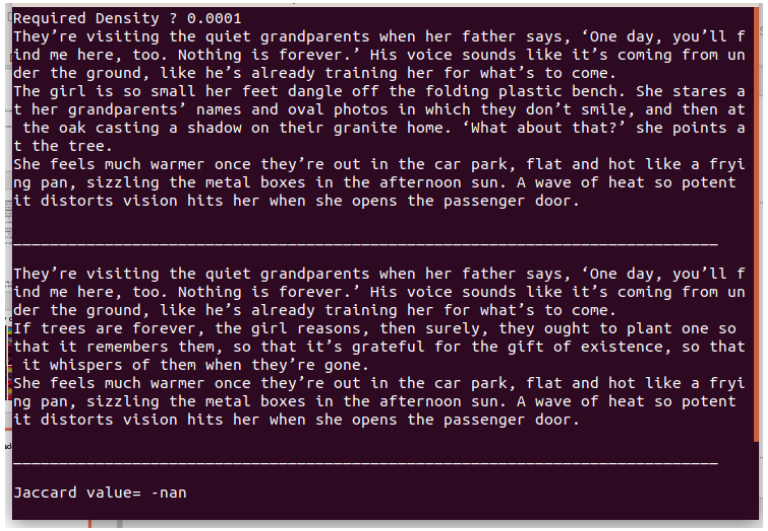
Larger k values (k=8) gives better matches

The Cost-Accuracy Tradeoff









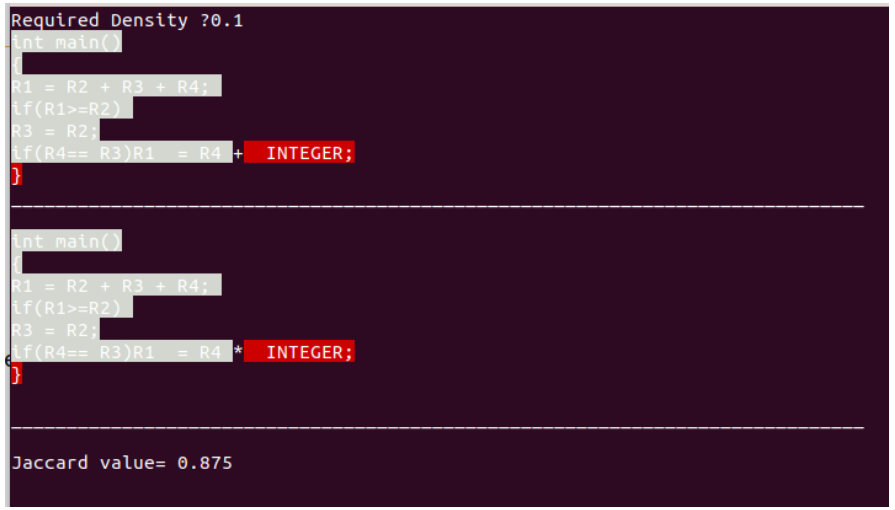
This demonstrates that as density decreases the smaller matches are undetected and finally all matches are lost Density values are (1, 0.1, 0.01, 0.005, 0.001 from top -bottom left-right). User has to choose the correct density as per his needs

Experiments (Masking)

```
int main()
{
alpha = beta + gamma  + delta;  /* add some numbers */
if(alpha>=beta)                 /* do some comparison */
gamma =    beta;
if(delta== gamma)
alpha  = delta *  50;
}
```

```
5
4
1 2
1 3
2 4
2 5
1
4
alpha 1
beta 2
gamma 3
delta 4
```

```
int main()
{
R1 = R2 + R3 + R4;
if(R1>=R2)
R3 = R2;
if(R4== R3)R1  = R4 *  INTEGER;
}
```



Improvements/future work

- Building it as a web service (underway)
- Using syncmers in place of minimizers as syncmers are shown to have better spacing distribution compared to minimizers.
- Improve the colour highlighting scheme to deal with overlaps in a better way.
- Currently the hasse diagram is directly taken from user however if a program uses many variables (say 100) it is difficult for the user to input the poset graph with 100 vertices