RUTGERS

UNIVERSITY | NEW BRUNSWICK

ECE 573 - Data Structures and Algorithms | Spring 2018

(Project# 11)

DNA Sequence Detector

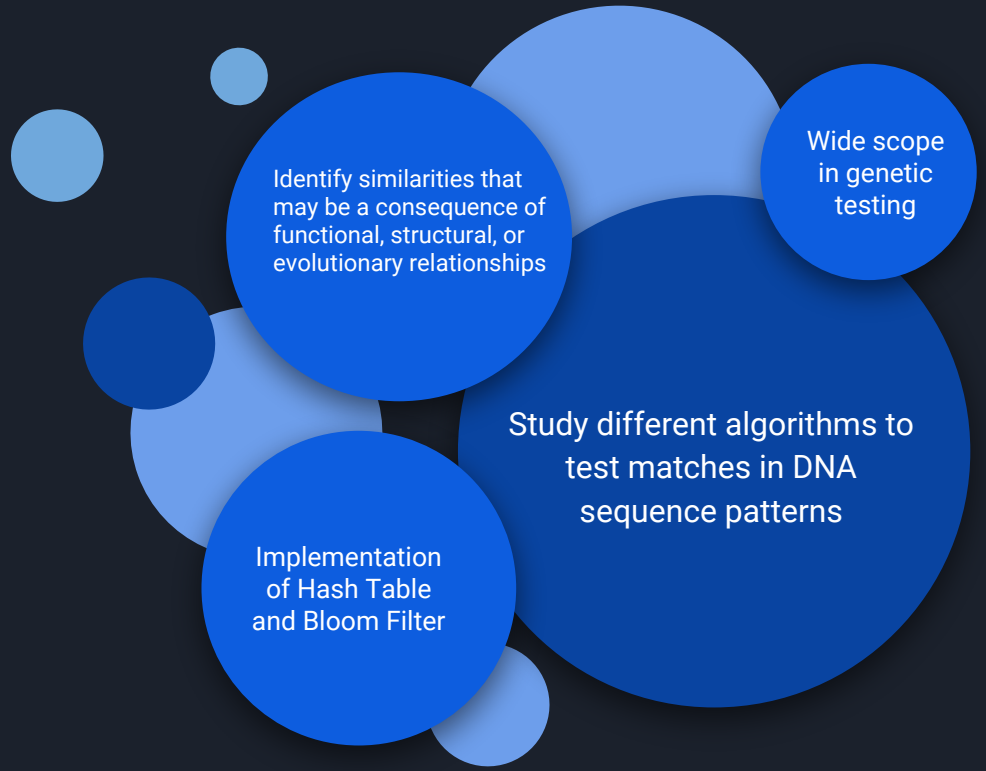
Divyaprakash Dhurandhar (dd839)

Tina Drew (tld95)

Anirudh Kulkarni (ak1512)

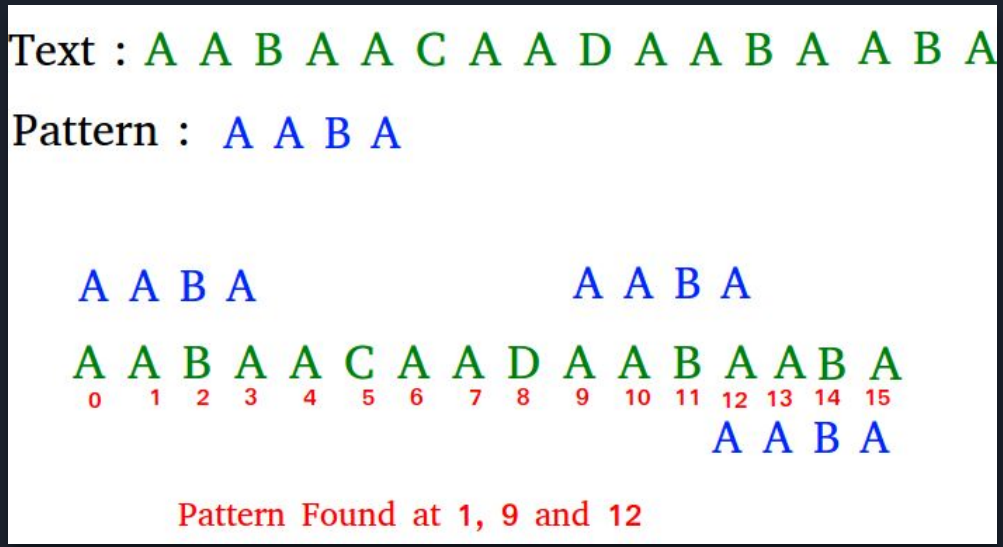
Juhi Tripathi (jt866)

Problem Statement - Motivation



Brute Force : Algorithm Design

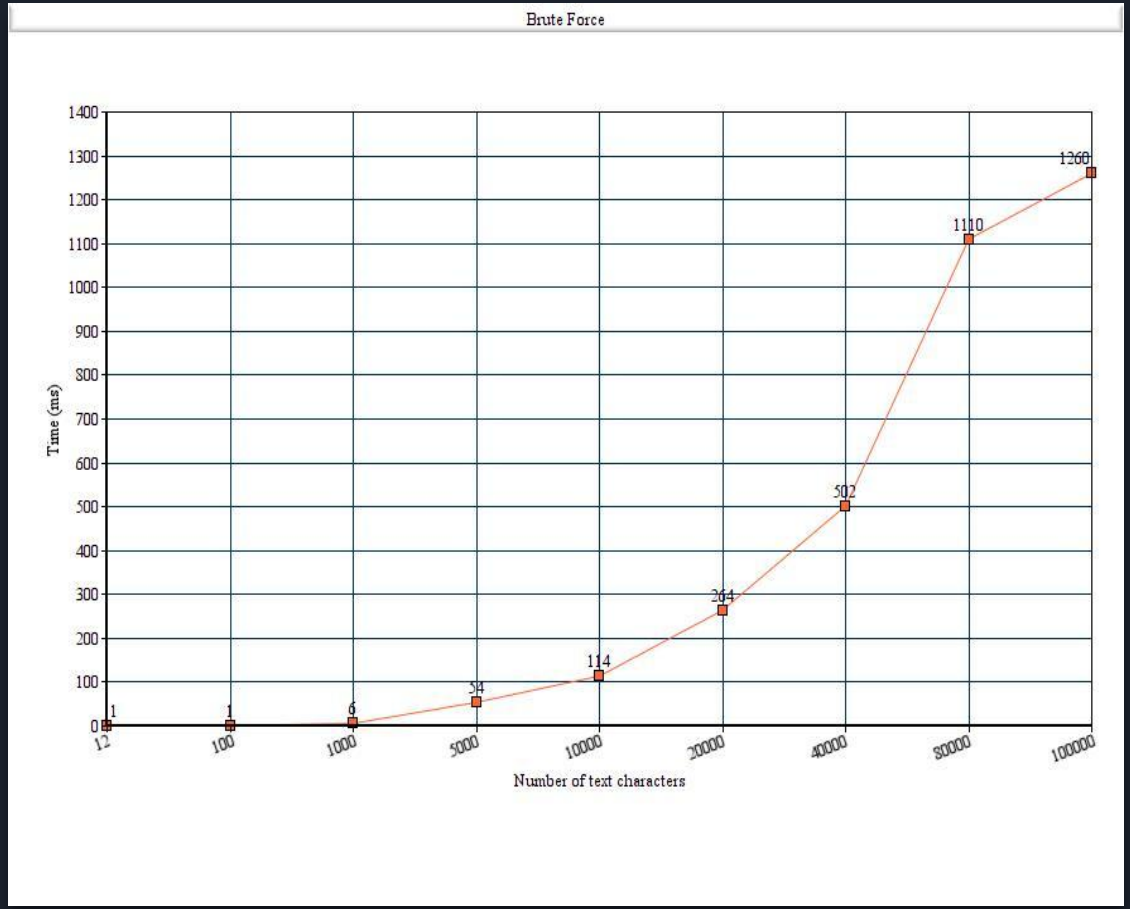
- Given a text *txt[0..n-1]* and a pattern *pat[0..m-1], the algorithm* uses a function *search(char pat[], char txt[])*that prints all occurrences of *pat[]* in *txt[]*, assuming that *n > m*.



Time Complexity

- **Best Case**
- **The best case occurs when the first character of the pattern is not present in the text at all.**
- **The number of comparisons in best case is O(n).**
- **Worst case**
- When all the characters in the pattern and the text are the same or the last character is different.
- Time complexity in worst case is - O(m*(n-m+1))

- Created randomly generated DNA sequences.
- Searched for a pattern “CTG” in the sequence.
- The graph resembles O(N^2)



Goldman Algorithm

Create Separate Chaining Hash Table

- Hash Table of optimum number of buckets ‘V’
- Constant hash function : *"weight of string % number of buckets"*
- Push the string to the hashed index

Comparison

- Compute hash function : *"weight of string % number of buckets"*
- Lookup in the hashtable at the hash mapping

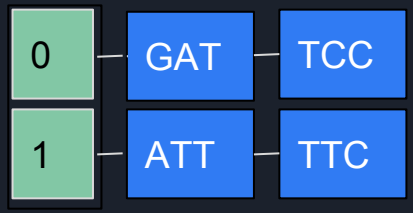
Goldman Algorithm

GATTCC : Corpus String

CATTCG : Pattern String

k = 3 and V = 2

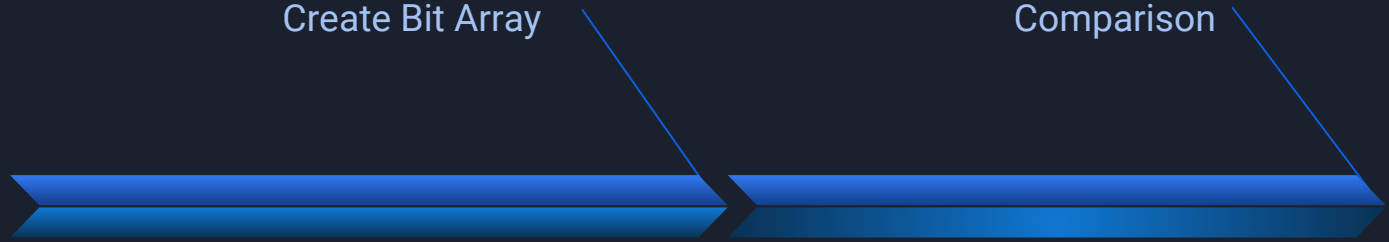
Possible k-length sequences :-
GAT
ATT
TTC
TCC



Possible k-length sequences :-	Lookup index in hash table :
CAT	0
ATT	1
TTC	1
TCG	0

Matches Found : ATT, TTC

Bloom Filter Implementation



- Bit array of optimum size ‘V’, initialized with 0
- Constant hash function : *"weight of string % number of buckets"*
- Set the hash index to 1 for every string

- Compute hash function : *"weight of string % number of buckets"*
- Lookup for the hashed index bit, in the array

  0 - not present

  1 - present

Bloom Filter Implementation

GATTCC : Corpus String

Possible k-length sequences :-	Corresponding index in array :-
GAT	0
ATT	1
TTC	1
TCC	0

k = 3 and V = 2

(initially)

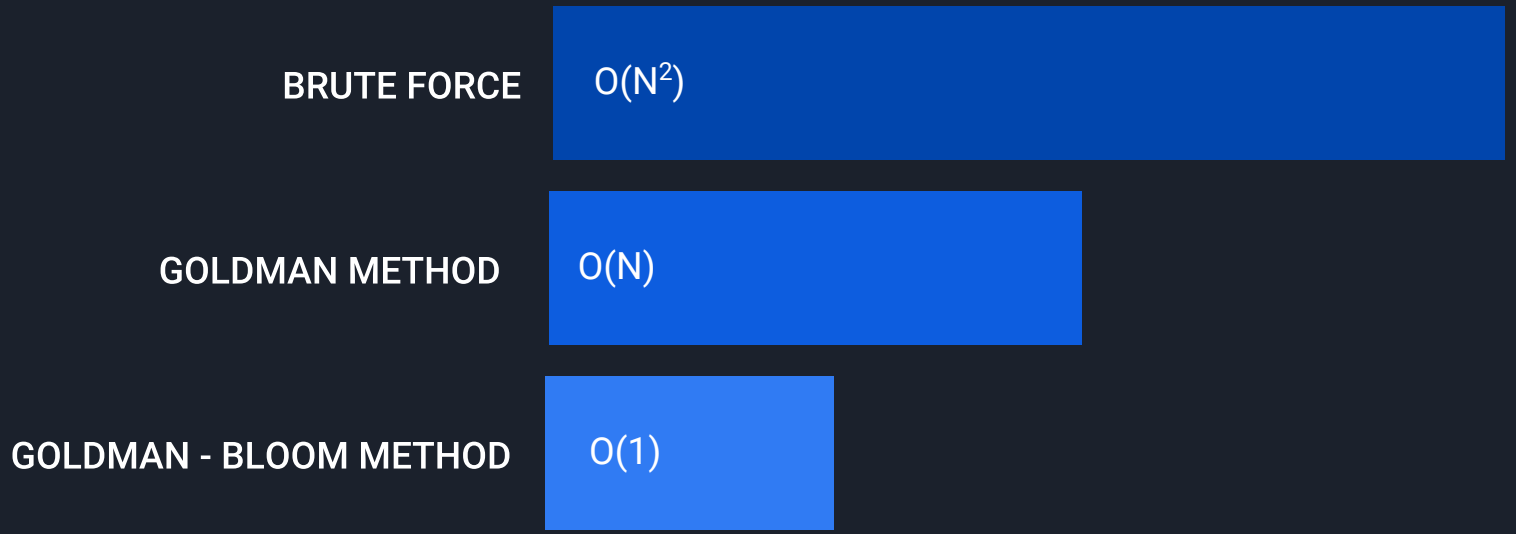
- GAT - sets index 0 to 1
- ATT - sets index 1 to 1
- TTC - index 1 is already 1
- TCC - index 0 is already 1

CATTCG : Pattern String

Possible k-length sequences :-	Lookup index in array :-
CAT	0
ATT	1
TTC	1
TCG	0

Matches Found : CAT, ATT, TTC, TCG
2 False positives! - Accounting for optimum value of ‘V’

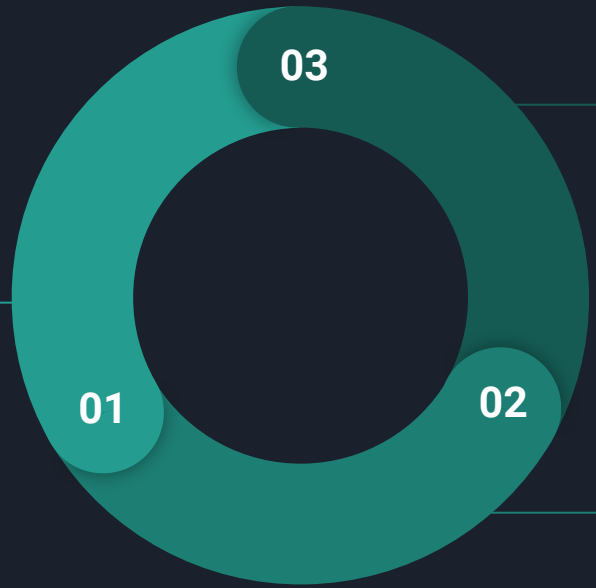
Complexity of Algorithms



Configuration and Details

Data Set

- Pattern File
- Corpus File
- Varying data size (12 to 10,000 characters)

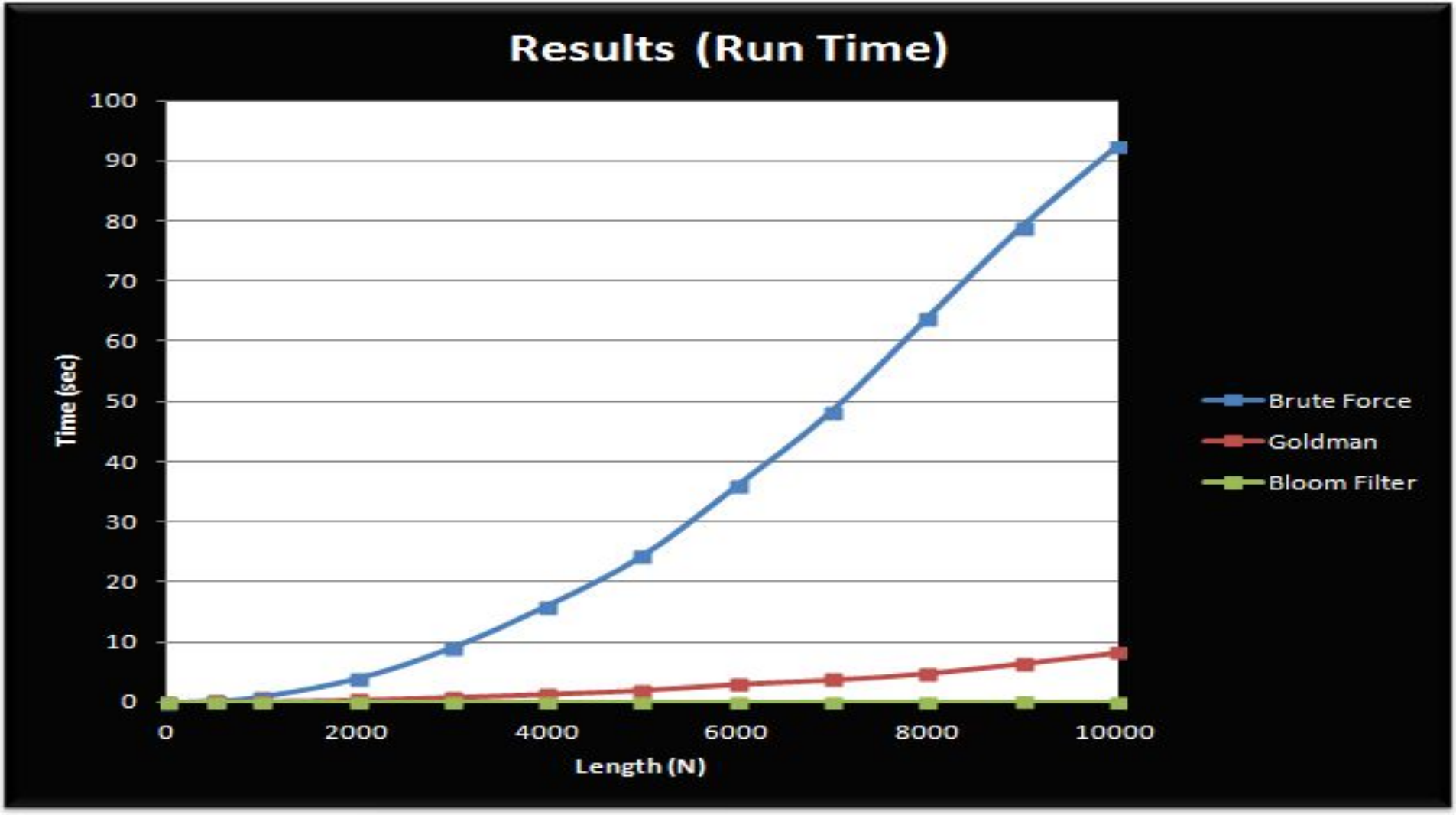


System

- **Operating System :** Windows 7
- **Processor :** Intel Core i3, 1.80 GHz
- **RAM:** 5 GB

Programming Languages

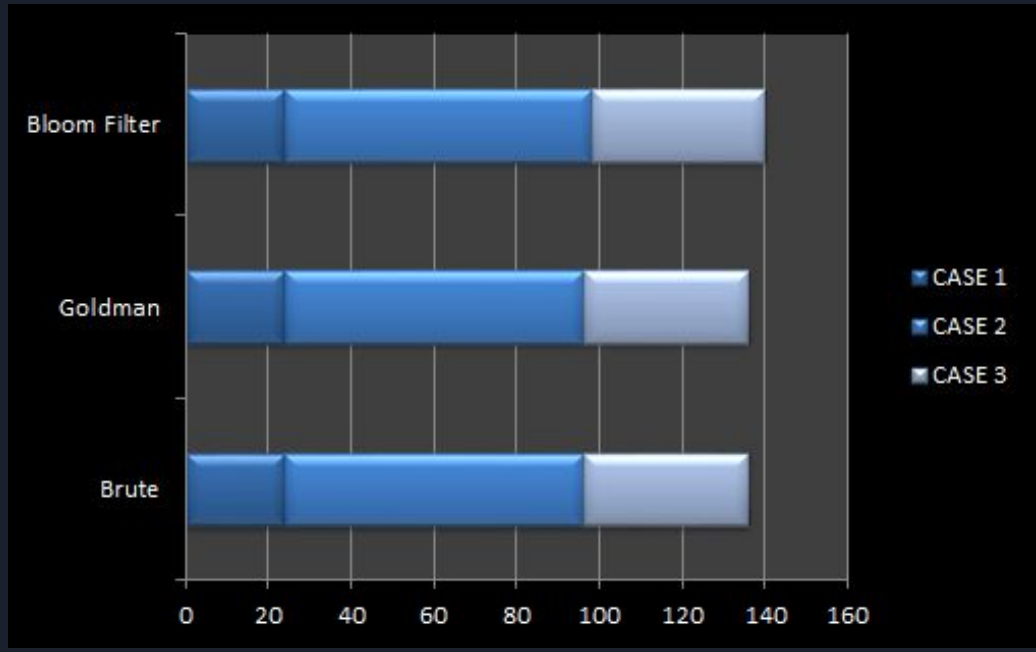
- Visual Studio 2017
- **Compiler** : MSVC
- Python 3.6.5


Results (Run Time)
Time (sec)
100
90
80
70
60
50
40
30
20
10
0
0
2000
4000
6000
8000
10000
Length (N)
Brute Force
Goldman
Bloom Filter

Results (Accuracy)

Number of Matches

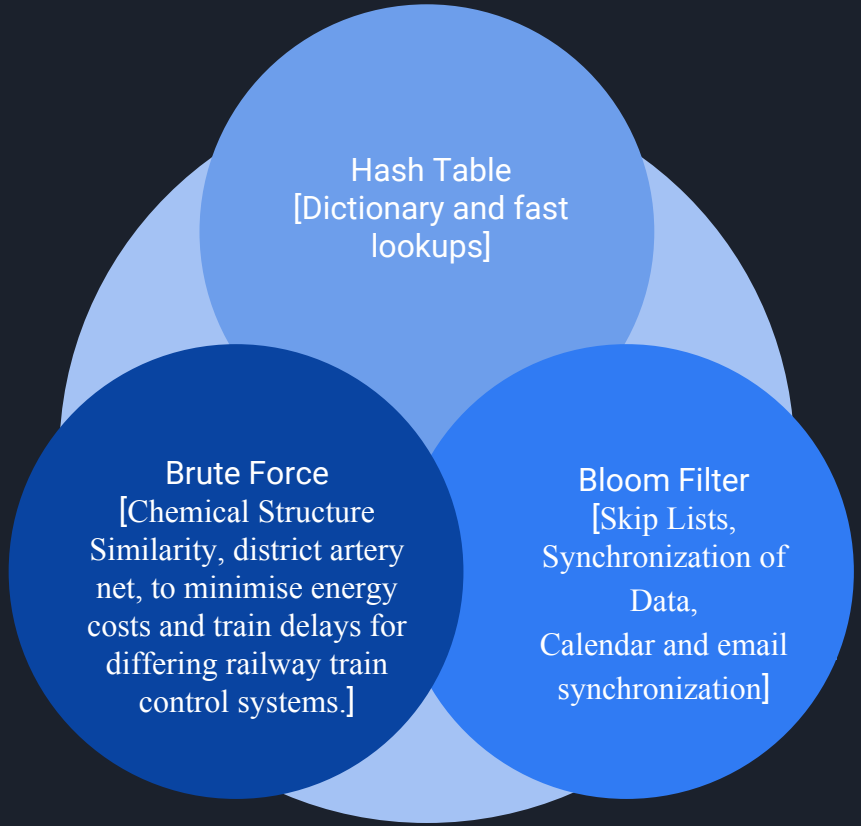
	CASE 1	CASE 2	CASE 3
Brute	24	72	40
Goldman	24	72	40
Bloom Filter	24	74	42



Conclusion

01	BRUTE FORCE	• Slowest Algorithm to implement Large Sequence
02	GOLDMAN ALGORITHM	• Faster technique with separate chaining hash table
03	GOLDMAN - BLOOM IMPLEMENTATION	• Reduce time and space complexity but lacks accuracy.

Applications



... Questions?

THANK YOU FOR YOUR ATTENTION!