Simulating the CVM Algorithm for Counting Distinct Elements in a Data Stream

By: Nader El-Ghotmi

Introduction

Introduction

What is the Count Distinct Problem?

Introduction

What is the Count Distinct Problem? Counting the number of distinct elements in a data stream.

Introduction

What is the Count Distinct Problem? Counting the number of distinct elements in a data stream.

The trivial solution:

Introduction

What is the Count Distinct Problem? Counting the number of distinct elements in a data stream.

The trivial solution: Store every distinct element in a data structure such as a hashmap.

Introduction

What is the Count Distinct Problem? Counting the number of distinct elements in a data stream.

The trivial solution: Store every distinct element in a data structure such as a hashmap.

Motivation:

Introduction

What is the Count Distinct Problem? Counting the number of distinct elements in a data stream.

The trivial solution: Store every distinct element in a data structure such as a hashmap.

Motivation: This solution scales proportionally to the number of distinct elements. Can we develop an algorithm with space which does not scale with the number of distinct elements?

The CVM Algorithm

Originally proposed by Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel

The CVM Algorithm

Originally proposed by Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel

Refined by Donald Knuth.

The CVM Algorithm

Originally proposed by Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel

Refined by Donald Knuth.

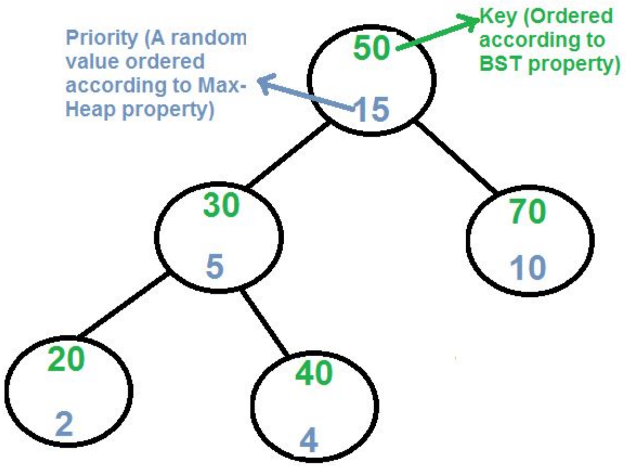
Provides an unbiased estimation for the number of distinct elements in a data stream.

Before we get started

Binary Search Tree + Max Heap

Keys sorted like BST, Values sorted like Max Heap

Get Max Value: O(1)
Search Key: O(logn)
Insert: O(logn)
Delete: O(logn)



Pseudocode for the CVM Algorithm

Pseudocode for the CVM Algorithm

Algorithm 1 CVM Algorithm

```
function CVM(A, s)
    Input: Data stream A = a_1, a_2, ..., a_n of size n,
  buffer size s
    t ← 0
    p ← 1
    B ← Empty Treap
    while t < n do
        t ← t + 1
        a ← a_t
        if a is in B then
            delete a from B
        end if
        u ← random number in [0, 1)
        if u ≥ p then
            NEXT
        else if |B| < s then
            Insert (a, u) into B
            NEXT
        else
            (a′, u′) ← (a, u) of max u in B
            if u > u′ then
                p ← u
            else
                Delete (a′, u′) from B
                Insert (a, u) into B
                p ← u′
            end if
        end if
    end while
    RETURN |B|/p
end function
```

Pseudocode for the CVM Algorithm

$$Pr(a_j \in B_t) = p_t,\ \forall j \in\ [1, t]$$

Algorithm 1 CVM Algorithm

```
function CVM(A, s)
    Input: Data stream A = a_1, a_2, ..., a_n of size n,
    buffer size s
    t ← 0
    p ← 1
    B ← Empty Treap
    while t < n do
        t ← t + 1
        a ← a_t
        if a is in B then
            delete a from B
        end if
        u ← random number in [0, 1)
        if u ≥ p then
            NEXT
        else if |B| < s then
            Insert (a, u) into B
            NEXT
        else
            (a′, u′) ← (a, u) of max u in B
            if u > u′ then
                p ← u
            else
                Delete (a′, u′) from B
                Insert (a, u) into B
                p ← u′
            end if
        end if
    end while
    RETURN |B|/p
end function
```

Pseudocode for the CVM Algorithm

$$Pr(a_j \in B_t) = p_t,\ \forall j \in\ [1, t]$$

$$E[|B_t|] = p_t \times |A_t|$$

Algorithm 1 CVM Algorithm

function CVM(A, s)
    Input: Data stream $A = a_1, a_2, ..., a_n$ of size n, buffer size s
    $t \leftarrow 0$
    $p \leftarrow 1$
    $B \leftarrow$ Empty Treap
    while $t < n$ **do**
        $t \leftarrow t + 1$
        $a \leftarrow a_t$
        if a is in B **then**
            delete a from B
        end if
        $u \leftarrow$ random number in $[0, 1)$
        if $u \geq p$ **then**
            NEXT
        else if $|B| < s$ **then**
            Insert (a, u) into B
            NEXT
        else
            $(a', u') \leftarrow (a, u)$ of max u in B
            if $u > u'$ **then**
                $p \leftarrow u$
            else
                Delete (a', u') from B
                Insert (a, u) into B
                $p \leftarrow u'$
            end if
        end if
    end while
    RETURN $|B|/p$
end function

Pseudocode for the CVM Algorithm

$$Pr(a_j \in B_t) = p_t,\ \forall j \in\ [1, t]$$

$$E[|B_t|] = p_t \times |A_t|$$

$$\frac{E[|B_t|]}{p_t} = |A_t|$$

Algorithm 1 CVM Algorithm

```
function CVM(A, s)
    Input: Data stream A = a_1, a_2, ..., a_n of size n,
    buffer size s
    t ← 0
    p ← 1
    B ← Empty Treap
    while t < n do
        t ← t + 1
        a ← a_t
        if a is in B then
            delete a from B
        end if
        u ← random number in [0, 1)
        if u ≥ p then
            NEXT
        else if |B| < s then
            Insert (a, u) into B
            NEXT
        else
            (a', u') ← (a, u) of max u in B
            if u > u' then
                p ← u
            else
                Delete (a', u') from B
                Insert (a, u) into B
                p ← u'
            end if
        end if
    end while
    RETURN |B|/p
end function
```

Time Complexity of the CVM Algorithm

Clearly we do n iterations for each element in the stream.

So O(nT(F(s))) where F(s) is what goes on inside each iteration.

Algorithm 1 CVM Algorithm

function CVM(A, s)
    Input: Data stream $A = a_1, a_2, ..., a_n$ of size n, buffer size s
    $t \leftarrow 0$
    $p \leftarrow 1$
    $B \leftarrow$ Empty Treap
    while $t < n$ **do**
        $t \leftarrow t + 1$
        $a \leftarrow a_t$
        if a is in B **then**
            delete a from B
        end if
        $u \leftarrow$ random number in $[0, 1)$
        if $u \geq p$ **then**
            NEXT
        else if $|B| < s$ **then**
            Insert (a, u) into B
            NEXT
        else
            $(a', u') \leftarrow (a, u)$ of max u in B
            if $u > u'$ **then**
                $p \leftarrow u$
            else
                Delete (a', u') from B
                Insert (a, u) into B
                $p \leftarrow u'$
            end if
        end if
    end while
    RETURN $|B|/p$
end function

Time Complexity of the CVM Algorithm

Algorithm 1 CVM Algorithm

```
function CVM(A, s)
    Input: Data stream A = a_1, a_2, ..., a_n of size n,
buffer size s
    t ← 0
    p ← 1
    B ← Empty Treap
    while t < n do
        t ← t + 1
        a ← a_t
        if a is in B then
            delete a from B
        end if
        u ← random number in [0, 1)
        if u ≥ p then
            NEXT
        else if |B| < s then
            Insert (a, u) into B
            NEXT
        else
            (a′, u′) ← (a, u) of max u in B
            if u > u′ then
                p ← u
            else
                Delete (a′, u′) from B
                Insert (a, u) into B
                p ← u′
            end if
        end if
    end while
    RETURN |B|/p
end function
```

Time Complexity of the CVM Algorithm

Time Complexity of the CVM Algorithm

$$T(F(s)) = \begin{cases} 2log(s) & \text{if } |B| \leq s \text{ and } a \in |B| \text{ and } u < p \\ log(s) & \text{if } |B| < s \text{ and } a \notin |B| \text{ and } u < p \\ log(s) & \text{if } |B| = s \text{ and } a \in |B| \text{ and } u \geq p \\ 2log(s) & \text{if } |B| = s \text{ and } a \notin |B| \text{ and } u < p \text{ and } u' \leq u \end{cases}$$

Time Complexity of the CVM Algorithm

$$T(F(s)) = \begin{cases} 2log(s) & \text{if } |B| \leq s \text{ and } a \in |B| \text{ and } u < p \\ log(s) & \text{if } |B| < s \text{ and } a \notin |B| \text{ and } u < p \\ log(s) & \text{if } |B| = s \text{ and } a \in |B| \text{ and } u \geq p \\ 2log(s) & \text{if } |B| = s \text{ and } a \notin |B| \text{ and } u < p \text{ and } u' \leq u \end{cases}$$

$$O(nlog(s))$$

Correctness of the CVM Algorithm

To prove it is a valid estimator we need to prove that the following property is held.

$$Pr(a_j \in B_t) = p_t,\ \forall j \in\ [1, t]$$

Algorithm 1 CVM Algorithm

```
function CVM(A, s)
    Input: Data stream A = a_1, a_2, ..., a_n of size n, buffer size s
    t ← 0
    p ← 1
    B ← Empty Treap
    while t < n do
        t ← t + 1
        a ← a_t
        if a is in B then
            delete a from B
        end if
        u ← random number in [0, 1)
        if u ≥ p then
            NEXT
        else if |B| < s then
            Insert (a, u) into B
            NEXT
        else
            (a′, u′) ← (a, u) of max u in B
            if u > u′ then
                p ← u
            else
                Delete (a′, u′) from B
                Insert (a, u) into B
                p ← u′
            end if
        end if
    end while
    RETURN |B|/p
end function
```

Correctness of the CVM Algorithm

To prove it is a valid estimator we need to prove that the following property is held.

$$Pr(a_j \in B_t) = p_t, \ \forall j \in \ [1, t]$$

Thus $\dfrac{E[|B_t|]}{p_t} = |A_t|$ is a valid estimation.

Algorithm 1 CVM Algorithm

```
function CVM(A, s)
    Input: Data stream A = a_1, a_2, ..., a_n of size n,
buffer size s
    t ← 0
    p ← 1
    B ← Empty Treap
    while t < n do
        t ← t + 1
        a ← a_t
        if a is in B then
            delete a from B
        end if
        u ← random number in [0, 1)
        if u ≥ p then
            NEXT
        else if |B| < s then
            Insert (a, u) into B
            NEXT
        else
            (a', u') ← (a, u) of max u in B
            if u > u' then
                p ← u
            else
                Delete (a', u') from B
                Insert (a, u) into B
                p ← u'
            end if
        end if
    end while
    RETURN |B|/p
end function
```

Correctness of the CVM Algorithm

To prove it is a valid estimator we need to prove that the following property is held.

$$Pr(a_j \in B_t) = p_t,\ \forall j \in \ [1, t]$$

Thus $\dfrac{E[|B_t|]}{p_t} = |A_t|$ is a valid estimation.

Unfortunately, we do not know the competitive ratio/upper bound on error.

Algorithm 1 CVM Algorithm

function CVM(A, s)
    Input: Data stream $A = a_1, a_2, ..., a_n$ of size n, buffer size s
    $t \leftarrow 0$
    $p \leftarrow 1$
    $B \leftarrow$ Empty Treap
    while $t < n$ **do**
        $t \leftarrow t + 1$
        $a \leftarrow a_t$
        if a is in B **then**
            delete a from B
        end if
        $u \leftarrow$ random number in $[0, 1)$
        if $u \geq p$ **then**
            NEXT
        else if $|B| < s$ **then**
            Insert (a, u) into B
            NEXT
        else
            $(a', u') \leftarrow (a, u)$ of max u in B
            if $u > u'$ **then**
                $p \leftarrow u$
            else
                Delete (a', u') from B
                Insert (a, u) into B
                $p \leftarrow u'$
            end if
        end if
    end while
    RETURN $|B|/p$
end function

Experimentation

Without a competitive ratio, experimentation can still provide us with an idea of what to expect.

Experimentation - Step 1

Generating data streams for testing

Stream 1: This data stream was designed by generating N random 7 digit integers.

Experimentation - Step 1

Generating data streams for testing

Experimentation - Step 1

Generating data streams for testing

Stream 1: This data stream was designed by generating N random 7 digit integers.

Stream 2: This stream counts up from 0 to N and applies mod 50000 to each integer. So 0-49999 repeating, which causes an even distribution of distinct integers in the range 0-49999.

Experimentation - Step 1

Generating data streams for testing

Stream 1: This data stream was designed by generating N random 7 digit integers.

Stream 2: This stream counts up from 0 to N and applies mod 50000 to each integer. So 0-49999 repeating, which causes an even distribution of distinct integers in the range 0-49999.

Stream 3: This stream simply counts up from 1 to N and thus only has distinct integers.

Experimentation - Step 1

Generating data streams for testing

Stream 1: This data stream was designed by generating N random 7 digit integers.

Stream 2: This stream counts up from 0 to N and applies mod 50000 to each integer. So 0-49999 repeating, which causes an even distribution of distinct integers in the range 0-49999.

Stream 3: This stream simply counts up from 1 to N and thus only has distinct integers.

Stream 4: This stream replaces every odd number with the same odd number (50003). As such approximately half of the stream is composed of distinct integers while the other half is the same element.

Experimentation - Step 1

Generating data streams for testing

Stream 1: This data stream was designed by generating N random 7 digit integers.

Stream 2: This stream counts up from 0 to N and applies mod 50000 to each integer. So 0-49999 repeating, which causes an even distribution of distinct integers in the range 0-49999.

Stream 3: This stream simply counts up from 1 to N and thus only has distinct integers.

Stream 4: This stream replaces every odd number with the same odd number (50003). As such approximately half of the stream is composed of distinct integers while the other half is the same element.

Stream 5: This stream follows a normal distribution in terms of the frequency of an integer being present in the stream. In other words, one integer will be present the most in the stream, two other integers will be present marginally less, and again for the next two integers.

Experimentation - Step 1

Generating data streams for testing

Stream 1: This data stream was designed by generating N random 7 digit integers.

Stream 2: This stream counts up from 0 to N and applies mod 50000 to each integer. So 0-49999 repeating, which causes an even distribution of distinct integers in the range 0-49999.

Stream 3: This stream simply counts up from 1 to N and thus only has distinct integers.

Stream 4: This stream replaces every odd number with the same odd number (50003). As such approximately half of the stream is composed of distinct integers while the other half is the same element.

Stream 5: This stream follows a normal distribution in terms of the frequency of an integer being present in the stream. In other words, one integer will be present the most in the stream, two other integers will be present marginally less, and again for the next two integers.

Stream 6: This stream is composed of the same integer repeated save for 0.1% of the stream which are unique values.

Experimentation - Step 2

Deciding which buffer sizes to use.

Experimentation - Step 2

Deciding which buffer sizes to use.

$$[1, 3, 7, 15, 28, 53, 95, 168, 291, 500]$$

Experimentation - Step 2

Deciding which buffer sizes to use.

$$[1, 3, 7, 15, 28, 53, 95, 168, 291, 500]$$

In the case of the analysis I used 1000 data points in between 1 and 1 million.

Experimentation - Step 3

Running the CVM Algorithm for every buffer size in the list on Stream 1.

Experimentation - Step 3

Running the CVM Algorithm for every buffer size in the list on Stream 1.

Experimentation - Step 3

Running the CVM Algorithm for every buffer size in the list on Stream 1.

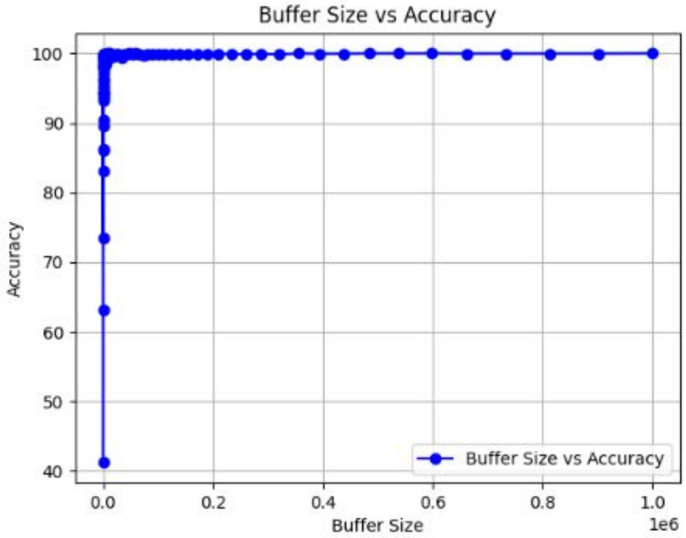


Figure 1: Buffer Size vs Accuracy for Stream 1 of size 1 million

Experimentation - Step 4

Truncating the charts to 30 data points.

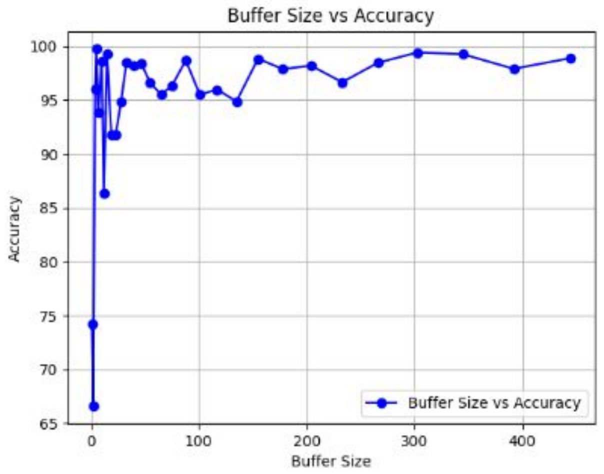


Figure 2: Buffer Size vs Accuracy for Stream 4 of size 1 million

Experimentation - Step 4

Truncating the charts to 30 data points.

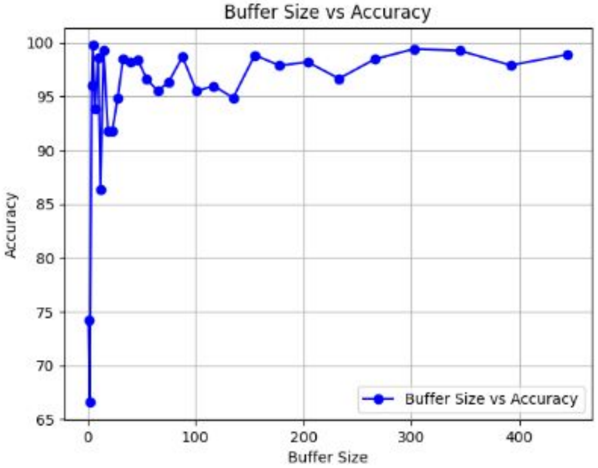


Figure 2: Buffer Size vs Accuracy for Stream 4 of size 1 million

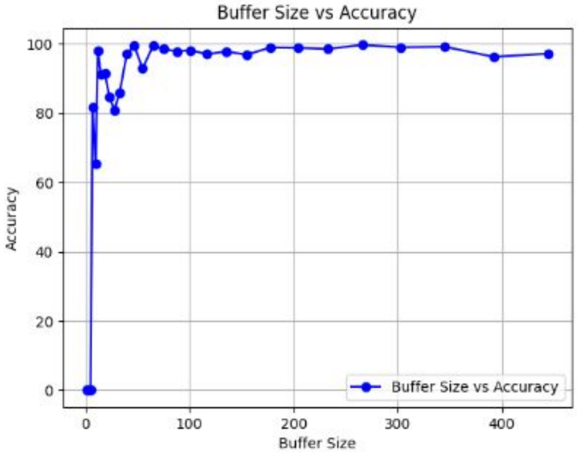


Figure 3: Buffer Size vs Accuracy for Stream 5 of size 1 million

Experimentation - Step 4

Truncating the charts to 30 data points.

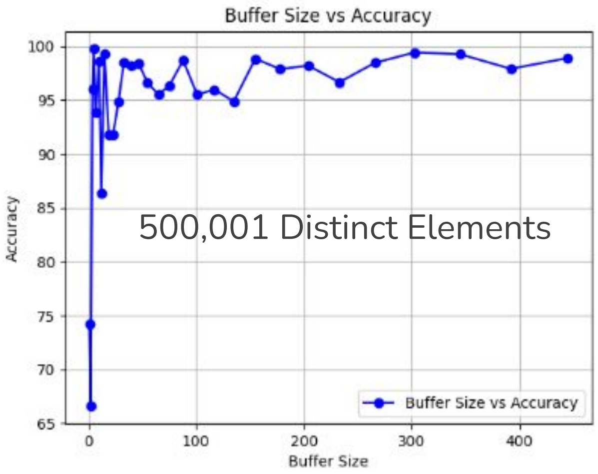


Figure 2: Buffer Size vs Accuracy for Stream 4 of size 1 million

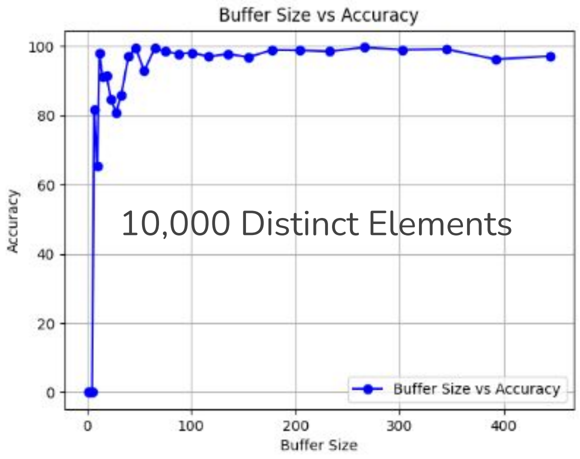


Figure 3: Buffer Size vs Accuracy for Stream 5 of size 1 million

Experimentation - Step 5

Comparing same streams on different stream sizes.

Experimentation - Step 5

Comparing same streams on different stream sizes.

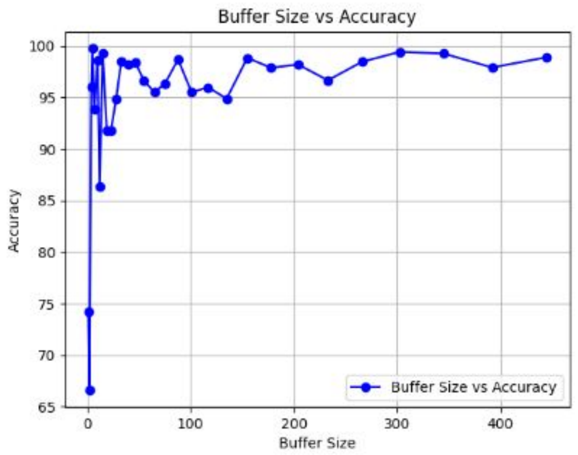


Figure 4: Buffer Size vs Accuracy for Stream 4 of size 1 million

Experimentation - Step 5

Comparing same streams on different stream sizes.

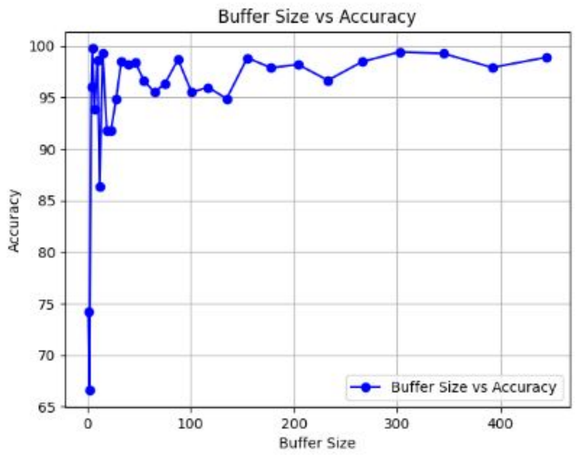


Figure 4: Buffer Size vs Accuracy for Stream 4 of size 1 million

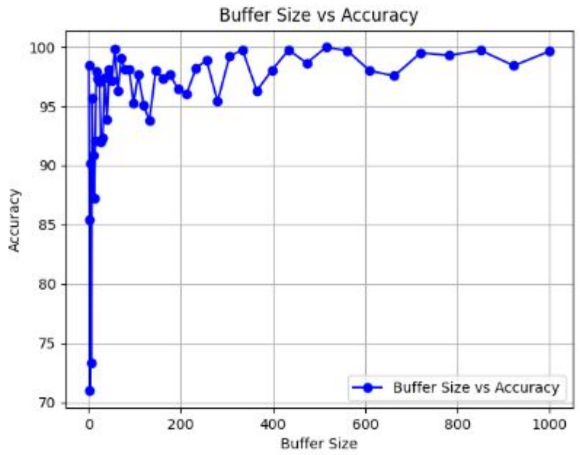


Figure 5: Buffer Size vs Accuracy for Stream 4 of size 10 million

Experimentation - Step 6

Checking the variance.

Experimentation - Step 6

Checking the variance.

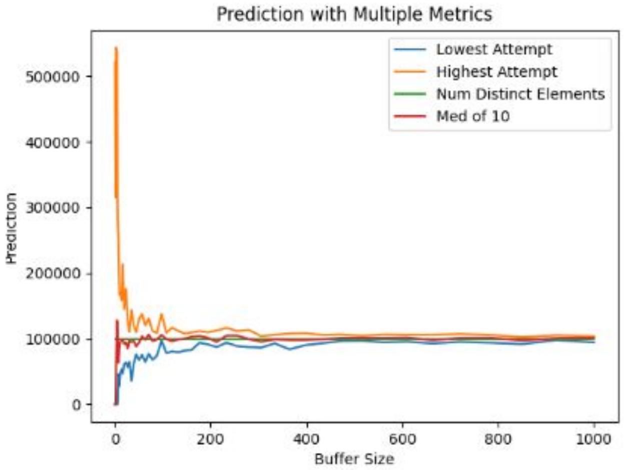


Figure 6: Buffer Size vs Best and Worst Case for Stream 5 of size 10 million

Experimentation - Step 6

Checking the variance.

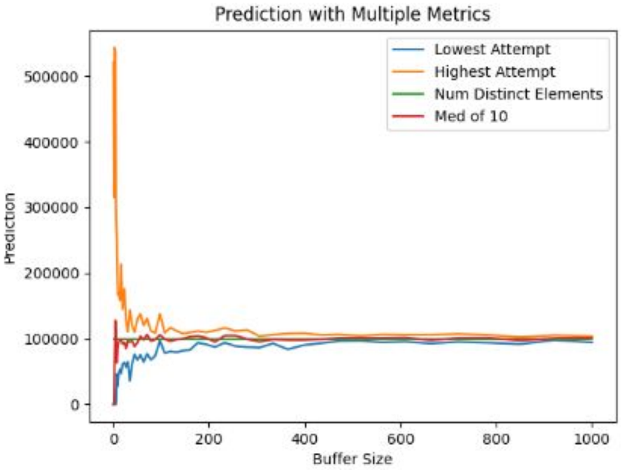


Figure 6: Buffer Size vs Best and Worst Case for Stream 5 of size 10 million

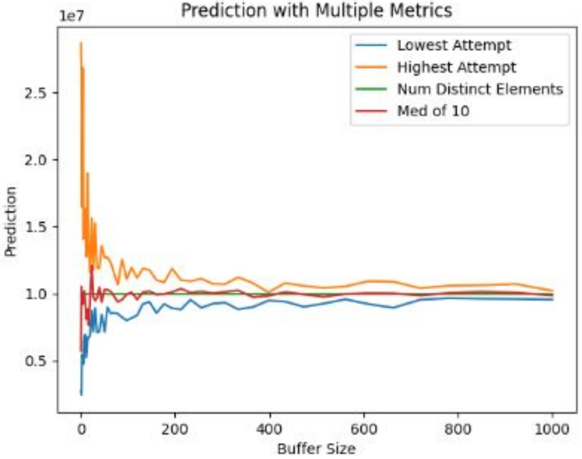


Figure 7: Buffer Size vs Best and Worst Case for Stream 3 of size 10 million

Experimentation - Step 6

Checking the variance.

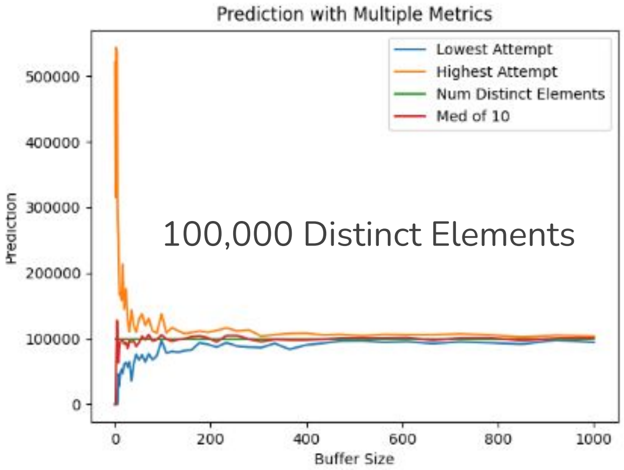


Figure 6: Buffer Size vs Best and Worst Case for Stream 5 of size 10 million

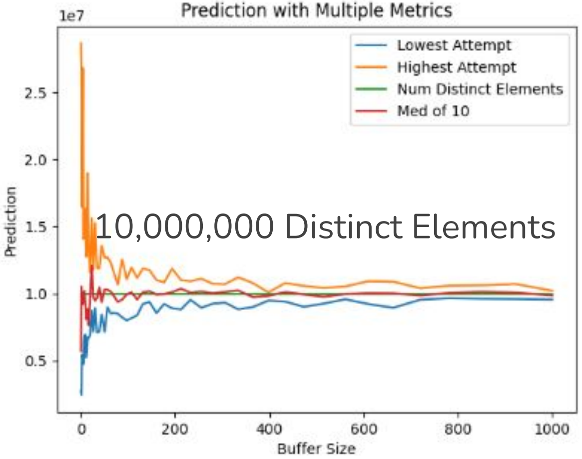


Figure 7: Buffer Size vs Best and Worst Case for Stream 3 of size 10 million

Conclusion

Summary: The CVM Algorithm provides a simple approach to estimating the number of distinct elements in a data stream. It lacks a proven competitive ratio, but useful information can be observed through experimentation.

Plans for the future:

- Implement in C++
- Develop a more rigorous plan for analysis
- Generate streams that mimic real world examples
- Try to define the competitive ratio

References

S. Chakraborty, N. V. Vinodchandran[1], and K. S. Meel. Distinct Elements in Streams: An Algorithm for the (Text) Book. In S. Chechik, G. Navarro, E. Rotenberg, and G. Herman, editors, 30th Annual European Symposium on Algorithms (ESA 2022), volume 244 of Leibniz International Proceedings in Informatics (LIPIcs), pages 34:1–34:6, Dagstuhl, Germany, 2022. Schloss Dagstuhl Leibniz-Zentrum fur Informatik.

D. Knuth. The CVM Algorithm for Estimating Distinct Elements in Streams, May 2023.

J. Leskovec, A. Rajaraman, and J. D. Ullman. Mining of massive datasets. Cambridge University Press, 2022.

Thank you!

Any questions?