

Probability and Computing – Streaming

Stefan Walzer | WS 2024/2025



1. Definition: What is a Streaming Algorithm?

2. Morris' Algorithm for $F_1 = m$

3. The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

4. Conclusion

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

What is a Streaming Algorithm?

- long input data stream $(a_1, \dots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \dots, a_m)$ with small relative error ε and failure probability δ .
↳ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store (a_1, \dots, a_m)).
↳ don't care about running time

Definition: What is a Streaming Algorithm?



Morris' Algorithm for $F_1 = m$



The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$



Conclusion



What is a Streaming Algorithm?

- long input data stream $(a_1, \dots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \dots, a_m)$ with small relative error ε and failure probability δ .
↪ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store (a_1, \dots, a_m)).
↪ don't care about running time

Formally, a streaming algorithm is given by three algorithms `init`, `update` and `result` used as follows:

```
Z ← init()
for i = 1 to m do
  Z ← update(Z, ai)
return result(Z)
```

Its space complexity is the space required for Z .

Definition: What is a Streaming Algorithm?



Morris' Algorithm for $F_1 = m$
○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$
○○○○○

Conclusion
○○○

What is a Streaming Algorithm?

- long input data stream $(a_1, \dots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \dots, a_m)$ with small relative error ε and failure probability δ .
↪ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store (a_1, \dots, a_m)).
↪ don't care about running time

Formally, a streaming algorithm is given by three algorithms `init`, `update` and `result` used as follows:

```
Z ← init()
for i = 1 to m do
  Z ← update(Z, ai)
return result(Z)
```

Its space complexity is the space required for Z .

Definition: What is a Streaming Algorithm?



Morris' Algorithm for $F_1 = m$
○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$
○○○○○

Conclusion
○○○

Today's Motivating Examples

- A Router approximately counts traffic over each connection.
↪ maybe: detect anomalies related to DDoS
- B Website approximately counts number of unique users visiting a resource.

What is a Streaming Algorithm?

- long input data stream $(a_1, \dots, a_m) \in [n]^m$ can only be read *once* from left to right
- goal: approximate some value $F = F(a_1, \dots, a_m)$ with small relative error ε and failure probability δ .
↪ streaming algorithms are approximation algorithms
- challenge: use less *space* than exact algorithm (in particular: cannot store (a_1, \dots, a_m)).
↪ don't care about running time

Formally, a streaming algorithm is given by three algorithms `init`, `update` and `result` used as follows:

```
Z ← init()
for i = 1 to m do
  Z ← update(Z, ai)
return result(Z)
```

Its space complexity is the space required for Z .

Definition: What is a Streaming Algorithm?



Morris' Algorithm for $F_1 = m$
○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$
○○○○○

Conclusion
○○○

Today's Motivating Examples

- A Router approximately counts traffic over each connection.
↪ maybe: detect anomalies related to DDoS
- B Website approximately counts number of unique users visiting a resource.

Today's Formal Results

- A Approximate $F_1(a_1, \dots, a_m) = m$ in expected space $\frac{1}{\varepsilon^2 \delta} \log \log m$.
- B Approximate $F_0(a_1, \dots, a_m) = |\{a_1, \dots, a_m\}|$ in expected space $\frac{1}{\varepsilon^2} \log(n) \cdot \log(m/\delta)$.

1. Definition: What is a Streaming Algorithm?

2. Morris' Algorithm for $F_1 = m$

3. The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

4. Conclusion

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

●○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

Attempt I: Naive Counting

Approximate Counting

- stream (a_1, \dots, a_m)
- want $F_1 = m$



Naive Counting

Algorithm init:

```
| Z ← 0  
| return Z
```

Algorithm update(Z, a):

```
| Z ← Z + 1  
| return Z
```

Algorithm result(Z):

```
| return Z
```

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

●○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

Attempt I: Naive Counting

Approximate Counting

- stream (a_1, \dots, a_m)
- want $F_1 = m$



Naive Counting

Algorithm init:

```
Z ← 0  
return Z
```

Algorithm update(Z, a):

```
Z ← Z + 1  
return Z
```

Algorithm result(Z):

```
return Z
```

Observations on Naive counting

- No errors ($\varepsilon = \delta = 0$).
- Requires $\lceil \log(m + 1) \rceil$ bits of memory.
- No *deterministic* algorithm can use less space

Attempt I: Naive Counting

Approximate Counting

- stream (a_1, \dots, a_m)
- want $F_1 = m$



Naive Counting

Algorithm init:

```
Z ← 0
return Z
```

Algorithm update(Z, a):

```
Z ← Z + 1
return Z
```

Algorithm result(Z):

```
return Z
```

Observations on Naive counting

- No errors ($\varepsilon = \delta = 0$).
- Requires $\lceil \log(m + 1) \rceil$ bits of memory.
- No *deterministic* algorithm can use less space
 - Would have to “reuse” a state Z .
 - Is then trapped in an infinite loop.
 - Result arbitrarily far off if m large enough.

Attempt II: Lossy Counting

Approximate Counting

- stream (a_1, \dots, a_m)
- want $F_1 = m$



Lossy Counting, parameter p

Algorithm init:

```
| Z ← 0  
| return Z
```

Algorithm update(Z, a):

```
| with probability  $p$  do  
| | Z ← Z + 1  
| return Z
```

Algorithm result(Z):

```
| return  $Z/p$ 
```

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

Attempt II: Lossy Counting

Approximate Counting

- stream (a_1, \dots, a_m)
- want $F_1 = m$



Lossy Counting, parameter p

Algorithm init:

```
Z ← 0  
return Z
```

Algorithm update(Z, a):

```
with probability  $p$  do  
  Z ← Z + 1  
return Z
```

Algorithm result(Z):

```
return  $Z/p$ 
```

Analysis (Exercise)

For any $p \in (0, 1]$ we have

- $\mathbb{E}[\text{result}] = m$
- $\Pr[|\text{result} - m| \leq \varepsilon m] \geq 1 - 2 \exp(-\varepsilon^2 pm/3)$.
- $\mathbb{E}[\text{space}] \leq \log_2(1 + mp) + 1$.

Attempt II: Lossy Counting

Approximate Counting

- stream (a_1, \dots, a_m)
- want $F_1 = m$



Lossy Counting, parameter p

Algorithm init:

```
Z ← 0
return Z
```

Algorithm update(Z, a):

```
with probability  $p$  do
  Z ← Z + 1
return Z
```

Algorithm result(Z):

```
return  $Z/p$ 
```

Analysis (Exercise)

For any $p \in (0, 1]$ we have

- $\mathbb{E}[\text{result}] = m$
- $\Pr[|\text{result} - m| \leq \varepsilon m] \geq 1 - 2 \exp(-\varepsilon^2 pm/3)$.
- $\mathbb{E}[\text{space}] \leq \log_2(1 + mp) + 1$.

Corollary

By choosing $p = \frac{3}{\varepsilon^2 m} \log(2/\delta)$ we get

$$\Pr[\text{fail}] \leq \delta \text{ and } \mathbb{E}[\text{space}] \leq \mathcal{O}(\log(\frac{1}{\varepsilon}) + \log \log(1/\delta)).$$

Attempt II: Lossy Counting

Approximate Counting

- stream (a_1, \dots, a_m)
- want $F_1 = m$



Lossy Counting, parameter p

Algorithm init:

```
Z ← 0
return Z
```

Algorithm update(Z, a):

```
with probability  $p$  do
  Z ← Z + 1
return Z
```

Algorithm result(Z):

```
return  $Z/p$ 
```

Analysis (Exercise)

For any $p \in (0, 1]$ we have

- $\mathbb{E}[\text{result}] = m$
- $\Pr[|\text{result} - m| \leq \epsilon m] \geq 1 - 2 \exp(-\epsilon^2 pm/3)$.
- $\mathbb{E}[\text{space}] \leq \log_2(1 + mp) + 1$.

Corollary

By choosing $p = \frac{3}{\epsilon^2 m} \log(2/\delta)$ we get

$$\Pr[\text{fail}] \leq \delta \text{ and } \mathbb{E}[\text{space}] \leq \mathcal{O}(\log(\frac{1}{\epsilon}) + \log \log(1/\delta)).$$

Serious Objection

Correctly choosing p requires already knowing m .
(or at least the order of magnitude of m)

- stream (a_1, \dots, a_m)
- want $F_1 = m$

Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:

```

| Z ← 0
| return Z

```

Algorithm update(Z, a):

```

| with probability  $2^{-Z}$  do
|   | Z ← Z + 1
| return Z

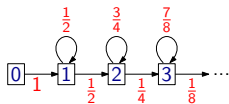
```

Algorithm result(Z):

```

| return  $2^Z - 1$ 

```



states of Z

transition probabilities

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

- stream (a_1, \dots, a_m)
- want $F_1 = m$

Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:

```

┌ Z ← 0
└ return Z
  
```

Algorithm update(Z, a):

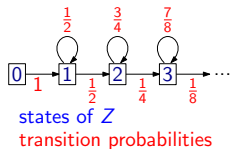
```

┌ with probability  $2^{-Z}$  do
  ┌ Z ← Z + 1
└ return Z
  
```

Algorithm result(Z):

```

┌ return  $2^Z - 1$ 
  
```



Definition: What is a Streaming Algorithm?

○

Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m.$

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

- stream (a_1, \dots, a_m)
- want $F_1 = m$

Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:

```

| Z ← 0
| return Z

```

Algorithm update(Z, a):

```

| with probability  $2^{-Z}$  do
|   | Z ← Z + 1
| return Z

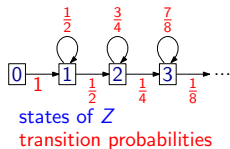
```

Algorithm result(Z):

```

| return  $2^Z - 1$ 

```



Definition: What is a Streaming Algorithm?

○

Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m.$

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.

Consider the expected change to 2^Z in one step...

- ... conditioned on a current value $j \in \mathbb{N}$:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j] = 2^{-j} \cdot (2^{j+1} - 2^j) + (1 - 2^{-j}) \cdot \underbrace{(2^j - 2^j)}_{=0} = 2 - 1 = 1.$$

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

○○○○○

Conclusion

○○○

- stream (a_1, \dots, a_m)
- want $F_1 = m$

Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:

```

┌ Z ← 0
└ return Z
  
```

Algorithm update(Z, a):

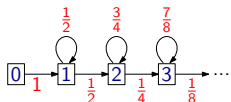
```

┌ with probability  $2^{-Z}$  do
  │ Z ← Z + 1
└ return Z
  
```

Algorithm result(Z):

```

└ return  $2^Z - 1$ 
  
```



states of Z

transition probabilities

Definition: What is a Streaming Algorithm?

○

Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m.$

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.

Consider the expected change to 2^Z in one step...

- ... conditioned on a current value $j \in \mathbb{N}$:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j] = 2^{-j} \cdot (2^{j+1} - 2^j) + (1 - 2^{-j}) \cdot \underbrace{(2^j - 2^j)}_{=0} = 2 - 1 = 1.$$

- ... unconditionally:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i}] \stackrel{\text{LTE}}{=} \sum_{j \geq 0} \Pr[Z_i = j] \cdot \underbrace{\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j]}_{=1} = \sum_{j \geq 0} \Pr[Z_i = j] = 1.$$

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

- stream (a_1, \dots, a_m)
- want $F_1 = m$

Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:

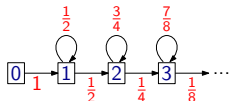
```
Z ← 0
return Z
```

Algorithm update(Z, a):

```
with probability  $2^{-Z}$  do
  Z ← Z + 1
return Z
```

Algorithm result(Z):

```
return  $2^Z - 1$ 
```



states of Z
transition probabilities

Definition: What is a Streaming Algorithm?

○

Lemma: Morris' Algorithm is an *Unbiased Estimator*

$\mathbb{E}[\text{result}] = m.$

Proof

Let Z_i for $i \in [m]$ denote the value of Z after i updates.

Consider the expected change to 2^Z in one step...

- ... conditioned on a current value $j \in \mathbb{N}$:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j] = 2^{-j} \cdot (2^{j+1} - 2^j) + (1 - 2^{-j}) \cdot \underbrace{(2^j - 2^j)}_{=0} = 2 - 1 = 1.$$

- ... unconditionally:

$$\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i}] \stackrel{\text{LTE}}{=} \sum_{j \geq 0} \Pr[Z_i = j] \cdot \underbrace{\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i} \mid Z_i = j]}_{=1} = \sum_{j \geq 0} \Pr[Z_i = j] = 1.$$

Hence:

$$\mathbb{E}[\text{result}] = \mathbb{E}[2^{Z_m} - 1] = \mathbb{E}[2^{Z_m} - 2^{Z_0}] = \mathbb{E}\left[\sum_{i=1}^m 2^{Z_{i+1}} - 2^{Z_i}\right] = \sum_{i=1}^m \underbrace{\mathbb{E}[2^{Z_{i+1}} - 2^{Z_i}]}_{=1} = m.$$

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

○○○○○

Conclusion

○○○

- stream (a_1, \dots, a_m)
- want $F_1 = m$

Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:

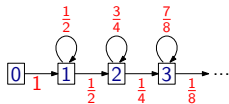
```
Z ← 0
return Z
```

Algorithm update(Z, a):

```
with probability  $2^{-Z}$  do
  Z ← Z + 1
return Z
```

Algorithm result(Z):

```
return  $2^Z - 1$ 
```



states of Z
transition probabilities

Lemma 1: Worryingly large Variance

$$\text{Var}(2^{Z_i}) = \frac{i^2 - i}{2} = \Theta(i^2).$$

Lemma 2

$$\mathbb{E}[2^{2Z_i}] = \frac{3i(i+1)}{2} + 1.$$

Proof of Lemma 1 using Lemma 2.

$$\begin{aligned} \text{Var}(2^{Z_i}) &= \mathbb{E}[2^{2Z_i}] - \mathbb{E}[2^{Z_i}]^2 \stackrel{\text{Lem. 2}}{=} \frac{3i(i+1)}{2} + 1 - (i+1)^2 \\ &= \frac{3i(i+1) + 2 - 2i^2 - 4i - 2}{2} = \frac{i^2 - i}{2}. \quad \square \end{aligned}$$

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

○○○○○

Conclusion

○○○

- stream (a_1, \dots, a_m)
- want $F_1 = m$

Attempt III: Morris' Algorithm

Morris' Algorithm

Algorithm init:

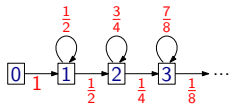
```
Z ← 0
return Z
```

Algorithm update(Z, a):

```
with probability  $2^{-Z}$  do
  Z ← Z + 1
return Z
```

Algorithm result(Z):

```
return  $2^Z - 1$ 
```



states of Z

transition probabilities

Lemma 1: Worryingly large Variance

$$\text{Var}(2^{Z_i}) = \frac{i^2 - i}{2} = \Theta(i^2).$$

Lemma 2

$$\mathbb{E}[2^{2Z_i}] = \frac{3i(i+1)}{2} + 1.$$

Proof of Lemma 2.

For $i \in \{0, 1\} \checkmark$. Let now $i \geq 1$. Note $\Pr[Z_{i+1} = 0] = \Pr[Z_i = 0] = 0$.

$$\begin{aligned} \mathbb{E}[2^{2Z_{i+1}}] &= \sum_{j \geq 1} 2^{2j} \Pr[Z_{i+1} = j] = \sum_{j \geq 1} 2^{2j} (\Pr[Z_i = j-1] \cdot 2^{-j+1} + \Pr[Z_i = j] \cdot (1 - 2^{-j})) \\ &= \sum_{j \geq 1} 2^{j+1} \Pr[Z_i = j-1] + \sum_{j \geq 1} 2^{2j} \Pr[Z_i = j] - \sum_{j \geq 1} 2^j \Pr[Z_i = j] \\ &= 4 \sum_{j \geq 0} 2^j \Pr[Z_i = j] + \sum_{j \geq 0} 2^{2j} \Pr[Z_i = j] - \sum_{j \geq 0} 2^j \Pr[Z_i = j] \\ &= 4\mathbb{E}[2^{Z_i}] + \mathbb{E}[2^{2Z_i}] - \mathbb{E}[2^{Z_i}] = 3\mathbb{E}[2^{Z_i}] + \mathbb{E}[2^{2Z_i}] = 3(i+1) + \mathbb{E}[2^{2Z_i}] \\ &\stackrel{\text{Ind.}}{=} 3(i+1) + \frac{3i(i+1)}{2} + 1 = \frac{3(i+2)(i+1)}{2} + 1. \quad \square \end{aligned}$$

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○●○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

○○○○○

Conclusion

○○○

Expected Space

$$\begin{aligned}\mathbb{E}[\text{space}] &\leq \mathbb{E}[\lceil \log_2(1 + Z_m) \rceil] \leq 1 + \mathbb{E}[\log_2(1 + Z_m)] = 1 + \mathbb{E}[\log_2(1 + \log_2(2^{Z_m}))] \\ &\stackrel{(*)}{\leq} 1 + \log_2(1 + \log_2(\mathbb{E}[2^{Z_m}])) = 1 + \log_2(1 + \log_2(m + 1)) = \Theta(\log \log m).\end{aligned}$$

(*) uses Jensen's inequality that you'll prove as an exercise.

Expected Space

$$\begin{aligned}\mathbb{E}[\text{space}] &\leq \mathbb{E}[\lceil \log_2(1 + Z_m) \rceil] \leq 1 + \mathbb{E}[\log_2(1 + Z_m)] = 1 + \mathbb{E}[\log_2(1 + \log_2(2^{Z_m}))] \\ &\stackrel{(*)}{\leq} 1 + \log_2(1 + \log_2(\mathbb{E}[2^{Z_m}])) = 1 + \log_2(1 + \log_2(m + 1)) = \Theta(\log \log m).\end{aligned}$$

(*) uses Jensen's inequality that you'll prove as an exercise.

Interim Conclusion: Morris is not good enough yet

- $\mathbb{E}[\text{result}] = m$ ✓ unbiased estimator
- $\mathbb{E}[\text{space}] = \mathcal{O}(\log \log m)$ ✓ highly space efficient
- $\text{Var}(\text{result}) = \Theta(m^2)$ ✗
 - Standarddeviation $\Theta(m)$
 \rightsquigarrow right order of magnitude, but not better.

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○●○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

○○○

Morris⁺: Use many copies of Morris' Algorithm

Theorem

Consider a streaming algorithm that maintains a sequence $Z = (Z_1, \dots, Z_s)$ of independent Morris-counters and returns $\text{result}(Z) := \frac{\text{result}(Z_1) + \dots + \text{result}(Z_s)}{s}$. For $s = \frac{1}{\varepsilon^2 \delta}$ we obtain

- $\mathbb{E}[\text{result}(Z)] = m$ and $\mathbb{E}[\text{space}] = \mathcal{O}\left(\frac{1}{\varepsilon^2 \delta} \log \log m\right)$
- $\Pr[|\text{result}(Z) - m| \leq \varepsilon m] = 1 - \mathcal{O}(\delta)$.

Morris⁺: Use many copies of Morris' Algorithm

Theorem

Consider a streaming algorithm that maintains a sequence $Z = (Z_1, \dots, Z_s)$ of independent Morris-counters and returns $\text{result}(Z) := \frac{\text{result}(Z_1) + \dots + \text{result}(Z_s)}{s}$. For $s = \frac{1}{\varepsilon^2 \delta}$ we obtain

- $\mathbb{E}[\text{result}(Z)] = m$ and $\mathbb{E}[\text{space}] = \mathcal{O}(\frac{1}{\varepsilon^2 \delta} \log \log m)$
- $\Pr[|\text{result}(Z) - m| \leq \varepsilon m] = 1 - \mathcal{O}(\delta)$.

Reminder on Variance

If X, Y are independent random variables and $s > 0$ then

- $\text{Var}(sX) = s^2 \text{Var}(X)$
- $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$

Proof of Concentration using Chebyshev

$$\begin{aligned}\text{Var}(\text{result}(Z)) &= \text{Var}\left(\frac{1}{s} \sum_{i=1}^s \text{result}(Z_i)\right) = \frac{1}{s^2} \text{Var}\left(\sum_{i=1}^s \text{result}(Z_i)\right) \\ &= \frac{1}{s^2} \sum_{i=1}^s \text{Var}(\text{result}(Z_i)) = \frac{s}{s^2} \text{Var}(\text{result}(Z_1)) = \frac{1}{s} \Theta(m^2) = \Theta(m^2/s).\end{aligned}$$

Chebyshev:

$$\Pr[X - \mathbb{E}[X] > c] \leq \frac{\text{Var}(X)}{c^2}.$$

$$\Pr[\text{fail}] = \Pr[|\text{result}(Z) - m| > \varepsilon m] = \Pr[|\text{result}(Z) - \mathbb{E}[\text{result}(Z)]| > \varepsilon m] \leq \frac{\text{Var}(\text{result}(Z))}{\varepsilon^2 m^2} = \Theta(1/(\varepsilon^2 s)) = \Theta(\delta). \quad \square$$

Morris*: Use a different base in Morris' Algorithm

Morris with base $1 < \rho \ll 2$

- In every update: increment Z with probability ρ^{-Z} .
- In the end: return $\rho^Z - 1$.

Modified Analysis

Show similarly to before:

- $\mathbb{E}[\text{result}] = m$
- $\text{Var}(\text{result}) = \Theta((\rho - 1)m^2)$

Choosing $\rho = 1 + \varepsilon^2 \delta$ gives:

- $\Pr[|\text{result} - m| > \varepsilon m] = \mathcal{O}(\delta)$.
- $\mathbb{E}[\text{space}] = \mathcal{O}(\log \log m + \log \frac{1}{\delta \varepsilon})$.

1. Definition: What is a Streaming Algorithm?

2. Morris' Algorithm for $F_1 = m$

3. The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

4. Conclusion

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

●○○○○

Conclusion

○○○

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

History

Remark: CVM is not well-known

Popular line of algorithms for F_0 by Philippe Flajolet et al:

- ~~1984: Flajolet-Martin~~ (deprecated)
 ↪ https://en.wikipedia.org/wiki/Flajolet-Martin_algorithm
- ~~2003: LogLog~~ (deprecated)
- 2007: HyperLogLog
 ↪ <https://en.wikipedia.org/wiki/HyperLogLog>

The CVM-Algorithm

- 2022: European Symposium on *Simplicity* in Algorithms 2022
 ↪ „Distinct Elements in Streams: An Algorithm for the (Text) Book“
- is a bit worse than HyperLogLog
- is easier to analyse than HyperLogLog

Next: We develop CVM in three steps.

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

●○○○

Conclusion

○○○

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

Attempt I: Naively storing the set

Naive Storing

Algorithm init:

```

| Z ← ∅
| return Z

```

Algorithm update(Z, a):

```

| Z ← Z ∪ {a}
| return Z

```

Algorithm result(Z):

```

| return |Z|

```

Observation

Naively storing the set requires $\Omega(F_0 \cdot \log n)$ bits.

Attempt II: Storing the set lossily

LossyStore, parameter p

Algorithm init:

```

┌  $Z \leftarrow \emptyset$ 
└ return  $Z$ 

```

Algorithm update(Z, a):

```

┌  $Z \leftarrow Z \setminus \{a\}$ 
  with probability  $p$  do
└  $Z \leftarrow Z \cup \{a\}$ 
  return  $Z$ 

```

Algorithm result(Z):

```

└ return  $|Z|/p$ ;

```

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

Attempt II: Storing the set lossily

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

LossyStore, parameter p

Algorithm init:

```

┌  $Z \leftarrow \emptyset$ 
└ return  $Z$ 

```

Algorithm update(Z, a):

```

┌  $Z \leftarrow Z \setminus \{a\}$ 
  with probability  $p$  do
└  $Z \leftarrow Z \cup \{a\}$ 
  return  $Z$ 

```

Algorithm result(Z):

```

└ return  $|Z|/p$ ;

```

Analysis

Let Z_0, \dots, Z_m be the states of Z over time. Invariant: Each $a \in \{a_1, \dots, a_i\}$ is in Z_i independently with probability p . Hence $|Z_m| \sim \text{Bin}(F_0, p)$.

Attempt II: Storing the set lossily

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

LossyStore, parameter p

Algorithm init:

```

| Z ← ∅
| return Z

```

Algorithm update(Z, a):

```

| Z ← Z \ {a}
| with probability p do
|   | Z ← Z ∪ {a}
| return Z

```

Algorithm result(Z):

```

| return |Z|/p;

```

Analysis

Let Z_0, \dots, Z_m be the states of Z over time. Invariant: Each $a \in \{a_1, \dots, a_i\}$ is in Z_i independently with probability p . Hence $|Z_m| \sim \text{Bin}(F_0, p)$.

- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
 \hookrightarrow result is *unbiased estimator* of F_0 .

Attempt II: Storing the set lossily

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

LossyStore, parameter p

Algorithm init:

```
Z ← ∅
return Z
```

Algorithm update(Z, a):

```
Z ← Z \ {a}
with probability p do
  Z ← Z ∪ {a}
return Z
```

Algorithm result(Z):

```
return |Z|/p;
```

Analysis

Let Z_0, \dots, Z_m be the states of Z over time. Invariant: Each $a \in \{a_1, \dots, a_i\}$ is in Z_i independently with probability p . Hence $|Z_m| \sim \text{Bin}(F_0, p)$.

- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
 \hookrightarrow result is *unbiased estimator* of F_0 .
- $\Pr[\text{fail}] = \Pr[|\text{result} - F_0| > \varepsilon F_0] = \Pr[||Z_m|/p - F_0| > \varepsilon F_0]$
 $= \Pr[||Z_m| - pF_0| > \varepsilon pF_0] = \Pr[||Z_m| - \mathbb{E}[|Z_m|]| > \varepsilon \mathbb{E}[|Z_m|]]$
 $\stackrel{\text{Chern.}}{\leq} 2 \exp(-\varepsilon^2 \mathbb{E}[|Z_m|]/3) = 2 \exp(-\varepsilon^2 pF_0/3)$.
 \hookrightarrow choose $p = p_\delta := \frac{3 \log(2/\delta)}{\varepsilon^2 F_0}$ for $\Pr[\text{fail}] \leq \delta$.

Chernoff for $X \sim \text{Bin}(n, p)$

$$\Pr[|X - \mathbb{E}[X]| > \varepsilon \mathbb{E}[X]] \leq 2 \exp(-\varepsilon^2 \mathbb{E}[X]/3).$$

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

○○○●○

Conclusion

○○○

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

Attempt II: Storing the set lossily

LossyStore, parameter p

Algorithm init:

```
Z ← ∅
return Z
```

Algorithm update(Z, a):

```
Z ← Z \ {a}
with probability p do
  Z ← Z ∪ {a}
return Z
```

Algorithm result(Z):

```
return |Z|/p;
```

Analysis

Let Z_0, \dots, Z_m be the states of Z over time. Invariant: Each $a \in \{a_1, \dots, a_i\}$ is in Z_i independently with probability p . Hence $|Z_m| \sim \text{Bin}(F_0, p)$.

- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
 \hookrightarrow result is *unbiased estimator* of F_0 .
- $\Pr[\text{fail}] = \Pr[|\text{result} - F_0| > \varepsilon F_0] = \Pr[||Z_m|/p - F_0| > \varepsilon F_0]$
 $= \Pr[||Z_m| - pF_0| > \varepsilon pF_0] = \Pr[||Z_m| - \mathbb{E}[|Z_m|]| > \varepsilon \mathbb{E}[|Z_m|]]$
 $\stackrel{\text{Chern.}}{\leq} 2 \exp(-\varepsilon^2 \mathbb{E}[|Z_m|]/3) = 2 \exp(-\varepsilon^2 pF_0/3)$.
 \hookrightarrow choose $p = p_\delta := \frac{3 \log(2/\delta)}{\varepsilon^2 F_0}$ for $\Pr[\text{fail}] \leq \delta$.
- Expected space *in the end* for $p = p_\delta$ ($\hat{\Delta} \neq$ peak space consumption)
 $\mathbb{E}[|Z_m| \cdot \mathcal{O}(\log n)] = F_0 p_\delta \cdot \mathcal{O}(\log n) = \mathcal{O}\left(\frac{\log(1/\delta)}{\varepsilon^2} \log n\right)$.

Chernoff for $X \sim \text{Bin}(n, p)$

$$\Pr[|X - \mathbb{E}[X]| > \varepsilon \mathbb{E}[X]] \leq 2 \exp(-\varepsilon^2 \mathbb{E}[X]/3).$$

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

○○○●○

Conclusion

○○○

Attempt II: Storing the set lossily

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

LossyStore, parameter p

Algorithm init:

```
Z ← ∅
return Z
```

Algorithm update(Z, a):

```
Z ← Z \ {a}
with probability p do
  Z ← Z ∪ {a}
return Z
```

Algorithm result(Z):

```
return |Z|/p;
```

Chernoff for $X \sim \text{Bin}(n, p)$

$$\Pr[|X - \mathbb{E}[X]| > \varepsilon \mathbb{E}[X]] \leq 2 \exp(-\varepsilon^2 \mathbb{E}[X]/3).$$

Analysis

Let Z_0, \dots, Z_m be the states of Z over time. Invariant: Each $a \in \{a_1, \dots, a_i\}$ is in Z_i independently with probability p . Hence $|Z_m| \sim \text{Bin}(F_0, p)$.

- $\mathbb{E}[\text{result}] = \mathbb{E}[|Z_m|/p] = \mathbb{E}[|Z_m|]/p = F_0 p/p = F_0$.
 \hookrightarrow result is *unbiased estimator* of F_0 .
- $\Pr[\text{fail}] = \Pr[|\text{result} - F_0| > \varepsilon F_0] = \Pr[||Z_m|/p - F_0| > \varepsilon F_0]$
 $= \Pr[||Z_m| - pF_0| > \varepsilon pF_0] = \Pr[||Z_m| - \mathbb{E}[|Z_m|]| > \varepsilon \mathbb{E}[|Z_m|]]$
 $\stackrel{\text{Chern.}}{\leq} 2 \exp(-\varepsilon^2 \mathbb{E}[|Z_m|]/3) = 2 \exp(-\varepsilon^2 pF_0/3)$.
 \hookrightarrow choose $p = p_\delta := \frac{3 \log(2/\delta)}{\varepsilon^2 F_0}$ for $\Pr[\text{fail}] \leq \delta$.
- Expected space *in the end* for $p = p_\delta$ ($\triangleleft \neq$ peak space consumption)
 $\mathbb{E}[|Z_m| \cdot \mathcal{O}(\log n)] = F_0 p_\delta \cdot \mathcal{O}(\log n) = \mathcal{O}\left(\frac{\log(1/\delta)}{\varepsilon^2} \log n\right)$.

Serious Objection: Need to know F_0 to choose p

- for $p \gg p_\delta$: space is wasted
- for $p \ll p_\delta$: failure becomes likely

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1, \dots, a_m\}|$

○○○●○

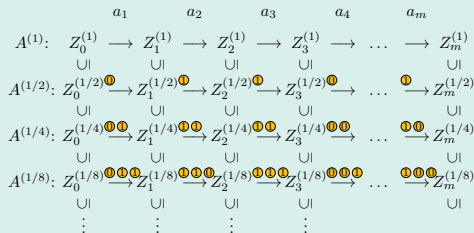
Conclusion

○○○

Attempt III: Adjust lossiness dynamically

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

Consider $A^{(p)} := \text{LossyStore}(p)$ with states $Z_0^{(p)}, \dots, Z_m^{(p)}$ for $p \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots\}$.



Coupling between executions of $A^{(p)}$

- $A^{(p/2)}$ uses coin tosses of $A^{(p)}$ and one more.
 “ $A^{(p/2)}$ keeps half of what $A^{(p)}$ keeps.”

Attempt III: Adjust lossiness dynamically

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

CVM, parameter T

Algorithm init:

```

Z ← ∅
P ← 1
return (P, Z)

```

Algorithm update($(P, Z), a$):

```

Z ← Z ∪ {a}
with probability P do
  Z ← Z ∪ {a}
while |Z| ≥ T do // shrink
  Z' ← ∅
  for a ∈ Z do
    with probability 1/2 do
      Z' ← Z' ∪ {a}
  (Z, P) ← (Z', P/2)
return (P, Z)

```

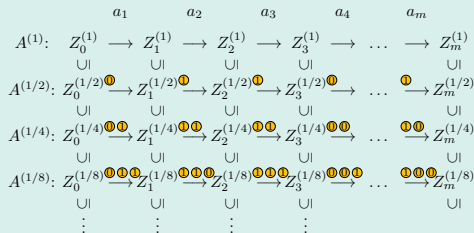
Algorithm result((P, Z)):

```

return |Z|/P

```

Consider $A^{(p)} := \text{LossyStore}(p)$ with states $Z_0^{(p)}, \dots, Z_m^{(p)}$ for $p \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots\}$.



Coupling between executions of $A^{(p)}$

- $A^{(p/2)}$ uses coin tosses of $A^{(p)}$ and one more.
 “ $A^{(p/2)}$ keeps half of what $A^{(p)}$ keeps.”

Attempt III: Adjust lossiness dynamically

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

CVM, parameter T

Algorithm init:

```
Z ← ∅
P ← 1
return (P, Z)
```

Algorithm update((P, Z), a):

```
Z ← Z ∪ {a}
with probability P do
  Z ← Z ∪ {a}
while |Z| ≥ T do // shrink
  Z' ← ∅
  for a ∈ Z do
    with probability 1/2 do
      Z' ← Z' ∪ {a}
  (Z, P) ← (Z', P/2)
return (P, Z)
```

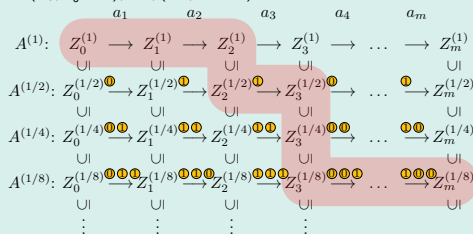
Algorithm result((P, Z)):

```
return |Z|/P
```

CVM behaves like LossyStore with dynamic p

Consider $A^{(p)} := \text{LossyStore}(p)$ with states $Z_0^{(p)}, \dots, Z_m^{(p)}$ for $p \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots\}$.

Let $(P_0, Z_0^{(\text{CVM})}), \dots, (P_m, Z_m^{(\text{CVM})})$ be the state of CVM.



Intuition: The path of CVM:

```
(x, y) ← (0, 0) // top left
for i = 1 to m do // m updates
  x ← x + 1 // go right
  while |Z_x^(2^-y)| ≥ T do
    y ← y + 1 // go down
final state is Z_m^(2^-y)
```

Coupling between executions of $A^{(p)}$ and CVM:

- $A^{(p/2)}$ uses coin tosses of $A^{(p)}$ and one more. "A^(p/2) keeps half of what A^(p) keeps."
- CVM uses coin tosses of $A^{(p)}$ to process elements.
- When shrinking, CVM inspects past coin tosses done by $A^{(p/2)}$. (the next unused coin for all $a \in Z$)

Effects of the coupling:

- $Z_j^{(\text{CVM})} = Z_j^{(P_j)}$ for $j \in [m]$
- $\text{result}^{(\text{CVM})} = \text{result}^{(P_m)}$
- $\text{fail}^{(\text{CVM})} = \text{fail}^{(P_m)}$

Attempt III: Adjust lossiness dynamically

- stream $(a_1, \dots, a_m) \in [n]^m$
- want $F_0 = |\{a_1, \dots, a_m\}|$

CVM, parameter T

Algorithm init:

```

Z ← ∅
P ← 1
return (P, Z)

```

Algorithm update((P, Z), a):

```

Z ← Z ∪ {a}
with probability P do
  Z ← Z ∪ {a}
while |Z| ≥ T do // shrink
  Z' ← ∅
  for a ∈ Z do
    with probability 1/2 do
      Z' ← Z' ∪ {a}
  (Z, P) ← (Z', P/2)
return (P, Z)

```

Algorithm result((P, Z):

```

return |Z|/P

```

Lemma: Failure Probability and Space

With $T = \frac{18 \log_2(2m/\delta)}{\epsilon^2}$ we get $\Pr[\text{fail}^{\text{CVM}}] = \mathcal{O}(\delta)$ and $\text{space}^{\text{CVM}} = \mathcal{O}\left(\frac{\log(m/\delta)}{\epsilon^2} \log n\right) + \lceil \log_2(\log_2(1/P_m)) \rceil$.

Analysis of CVM's failure probability (a bit sketchy)

- Recall: LossyStore($p_\delta = \frac{3 \log(2/\delta)}{\epsilon^2 F_0}$) has failure probability $\leq \delta$. Assume p_δ is power of 2.
- Then $\Pr[\text{fail}^{(p_\delta)}] \leq \delta$, $\Pr[\text{fail}^{(2p_\delta)}] \leq \delta^2$, $\Pr[\text{fail}^{(4p_\delta)}] \leq \delta^4, \dots$
- Therefore $\Pr[\text{fail}^{(1)}] + \dots + \Pr[\text{fail}^{(2p_\delta)}] + \Pr[\text{fail}^{(p_\delta)}] \leq \dots + \delta^8 + \delta^4 + \delta^2 + \delta = \mathcal{O}(\delta)$.

$$\begin{aligned}
 \Pr[P_m < p_\delta] &= \Pr[|Z_j^{(p_\delta)}| \geq T \text{ for some } j \in [m]] \leq m \cdot \Pr[|Z_m^{(p_\delta)}| \geq T] \\
 &= m \cdot \Pr_{Z \sim \text{Bin}(F_0, p_\delta)}[Z \geq T] \stackrel{\Delta}{=} m \cdot 2^{-T} \leq m \cdot 2^{-\log(m/\delta)} = \delta.
 \end{aligned}$$

where Δ uses a Chernoff bound and $6\mathbb{E}[Z] = 6F_0 p_\delta = \frac{18 \log_2(2/\delta)}{\epsilon^2} \leq T$.

- $\text{fail}^{\text{CVM}} \Leftrightarrow \text{fail}^{(P_m)} \Rightarrow (P_m < p_\delta \vee \text{fail}^{(1)} \vee \text{fail}^{(2)} \vee \dots \vee \text{fail}^{(p_\delta)})$

$$\text{Finally: } \Pr[\text{fail}^{\text{CVM}}] \leq \Pr[P_m < p_\delta \vee \text{fail}^{(1)} \vee \text{fail}^{(2)} \vee \dots \vee \text{fail}^{(p_\delta)}] \stackrel{\text{UB}}{\leq} \delta + \mathcal{O}(\delta) = \mathcal{O}(\delta).$$

Streaming Algorithms

- Input read only once, from left to right.
- Goal: Use little space. (less than what is needed to store input stream)
- Motivation: Network actor wants to maintain statistic on traffic.

Morris⁺ Algorithm for Counting the Stream Length

- approximation in space $\mathcal{O}\left(\frac{1}{\varepsilon^2 \delta} \log \log m\right)$ // or $\mathcal{O}(\log \log m + \log \frac{1}{\delta \varepsilon})$ using Morris*?
(ε = relative error, δ = failure probability)
- deterministic algorithms need space $\lceil \log(1 + m) \rceil$

CVM Algorithm for Counting *Distinct* Elements

- approximation in space $\mathcal{O}\left(\frac{1}{\varepsilon^2} \log(n) \log(m/\delta)\right)$

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

●○○

- Definition Streamingalgorithmen:
 - Was ist die Aufgabe eines Streamingalgorithmus (in Bezug auf eine Größe $F = F(a_1, \dots, a_m)$)?
 - Was ist die spezifische Herausforderung für Streamingalgorithmen?
- Streamingalgorithmen für $F_1 = m$:
 - Was könnte ein Anwendungsfall sein, in dem man F_1 schätzen möchte?
 - Wie viel Speicher braucht man wenn man einfach nur zählt? Kann ein deterministischer Algorithmus etwas Schlaures machen?
 - Wie funktioniert der LossyCounting Algorithmus? Warum hilft dieser uns nicht weiter?
 - Wie funktioniert Morris' Algorithmus?
 - Beweise, dass Morris' Algorithmus erwartungstreu ist.*
 - Beweise, dass der Speicherbedarf von Morris doppelt logarithmisch in m ist.
 - Welche Schwäche hatte Morris' Algorithmus noch und wie haben wir diese behoben?

Definition: What is a Streaming Algorithm?

○

Morris' Algorithm for $F_1 = m$

○○○○○○○

The CVM Algorithm for $F_0 = |\{a_1 \dots, a_m\}|$

○○○○○

Conclusion

●●●

- Streamingalgorithmen für $F_0 = \{a_1, \dots, a_m\}$:
 - Was könnte ein Anwendungsfall sein, in dem man F_0 schätzen möchte?
 - Wie viel Speicher braucht der naive deterministische Algorithmus? Was können wir mit CVM erreichen?
 - Als Zwischenschritt haben wir den Algorithmus LossyStore formuliert. Wie funktioniert dieser?
 - Wie funktioniert der CVM Algorithmus? Wie steht dieser mit dem LossyStore Algorithmus in Verbindung?
 - In der Analyse der Fehlerwahrscheinlichkeit von CVM haben wir zwei Arten von Problemen unterschieden. Welche?*