Identifying Optimal Parameters for Approximate Randomized Algorithms

Vimuth Fernando, Keyur Joshi, Darko Marinov, Sasa Misailovic
University of Illinois at Urbana-Champaign

WAX 2019

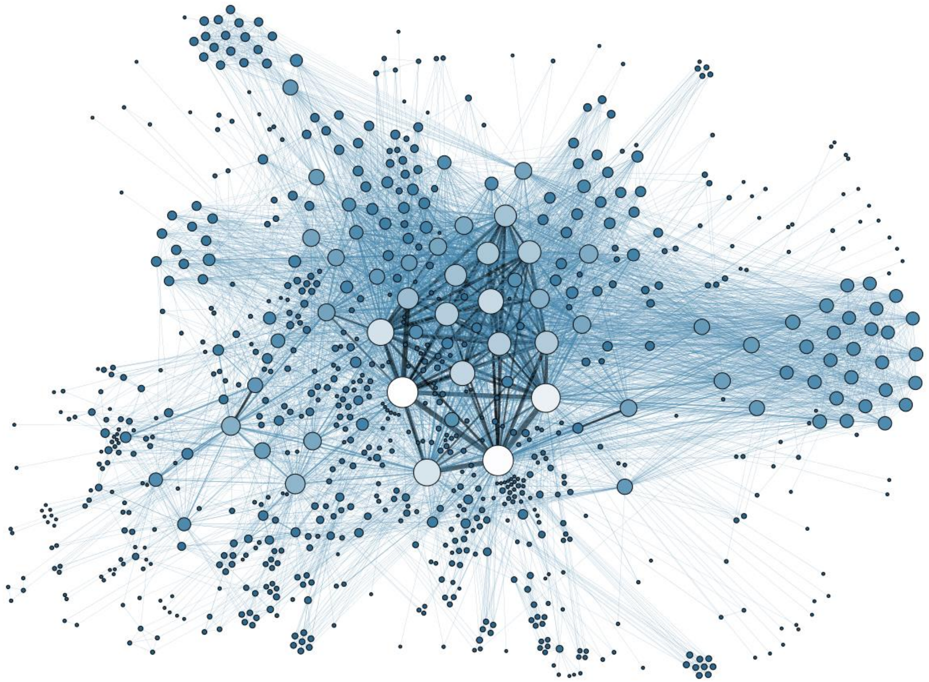
June 22, 2019, Phoenix, Arizona

Randomized Approximate Algorithms

Modern applications deal with large amounts of data

Obtaining exact answers for such applications is resource intensive

Randomized Approximate algorithms give a "good enough" answer in a much more efficient manner



Randomized Approximate Algorithms

Used in many domains

- HyperLogLog, Bloom filter - Data analytics
- Approximate matrix multiplication - Numerical linear algebra
- Locality sensitive hashing – Fingerpriting multimedia

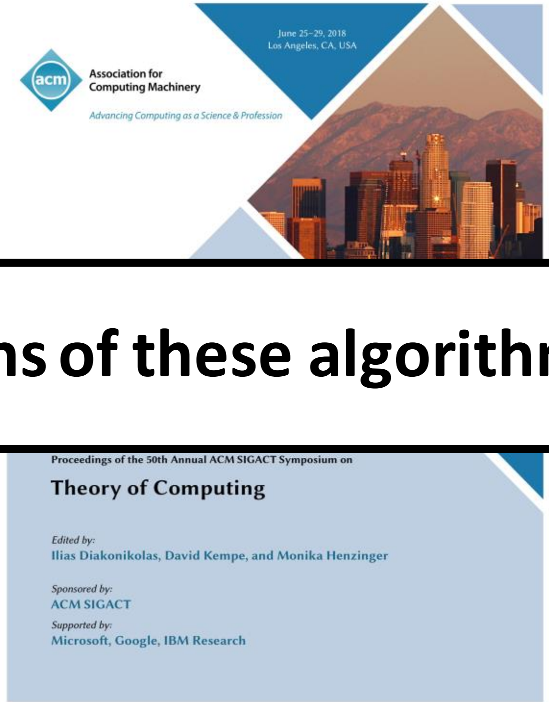
Often sub-linear in space/ runtime

Come with analytically derived specifications of accuracy/performance.

- e.g., an algorithm will have small errors with high probability

Randomized Approximate Algorithms

Randomized approximate algorithms have attracted the attention of many authors and researchers



Developers struggle to properly test/optimize implementations of these algorithms

Example: Count-min Sketch

- Count the **frequency of unique elements** in a large data set using sub-linear space
- Provides an estimate of the frequency with a bounded error

Data set: **x** **y** **z** **x** **…** **x**

x	:	95
y	:	135
z	:	935

Example: Count-min Sketch

- Count the frequency of unique elements in a data set using sub-linear space
- Use (h * w) counters

$hash_1$	0	0	0	0	…	0
$hash_2$	0	0	0	0	…	0
…	0	0	0	0	…	0
$hash_h$	0	0	0	0	…	0

h

w

Example: Count-min Sketch

- Count the frequency of unique elements in a data set using sub-linear space

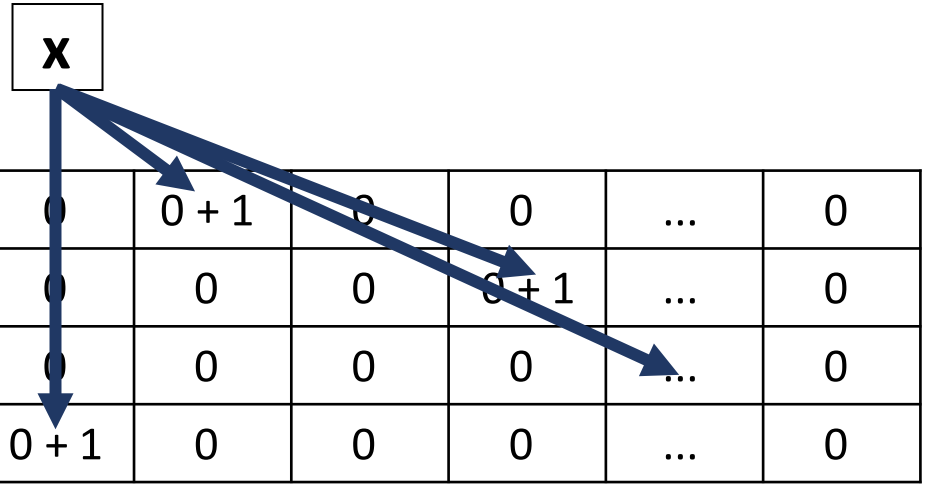
Data set: **x**

$hash_1$	0	0	0	0	...	0
$hash_2$	0	0	0	0	...	0
...	0	0	0	0	...	0
$hash_h$	0	0	0	0	...	0

Example: Count-min Sketch

- Count the frequency of unique elements in a data set using sub-linear space

Data set: **x**



$hash_1$	0	0 + 1	0	0	...	0
$hash_2$	0	0	0	0 + 1	...	0
...	0	0	0	0	...	0
$hash_h$	0 + 1	0	0	0	...	0

Example: Count-min Sketch

- Count the frequency of unique elements in a data set using sub-linear space

Data set: **x** **y**

$hash_1$	0 + 1	1	0	0	...	0
$hash_2$	0	0	0	1 + 1	...	0
...	0	0	0	0	...	0
$hash_h$	1	0	0 + 1	0	...	0

Example: Count-min Sketch

- Count the frequency of unique elements in a data set using sub-linear space

Data set: | **x** | **y** | **z** | **x** | **…** | **x** |

$hash_1$	50	200	12	454	…	64
$hash_2$	12	213	21	132	…	7657
…	49	842	12	23	…	67
$hash_h$	343	5	121	23	…	435

Example: Count-min Sketch

- Count the frequency of unique elements in a data set using sub-linear space

Data set: **x** **y** **z** **x** **…** **y**

$hash_1$	50	200	12	454	…	64
$hash_2$	12	213	21	132	…	7657
…	49	842	12	23	…	67
$hash_h$	343	5	121	23	…	435

Query: **x**

Example: Count-min Sketch

- Count the frequency of unique elements in a data set using sub-linear space

Data set: | x | y | z | x | … | y |

$hash_1$	50	200	12	454	…	64
$hash_2$	12	213	21	132	…	7657
…	49	842	12	23	…	67
$hash_h$	343	5	121	23	…	435

Query: x

Estimate count: min(343, 200, 132, …) = 132

Count-min Sketch Accuracy Specification*

Correctness Guarantee:

$$P\,[\,error < N\ *\epsilon] > 1\ -\ \delta$$

- $error$ $-$ difference between estimate and actual count
- $N -$ size of the data set
- Number of hash functions (h) and the number of bins per hash (w) is set using the values for ϵ and δ

$$w = \lceil e/\epsilon \rceil,\ h = \lceil \ln(1/\delta) \rceil$$

*G. Cormode and S. Muthukrishnan, “An improved data stream summary: the Count-Min sketch and its applications,” Journal of Algorithms, vol. 55, 2005

AxProf: Algorithmic Profiling for Randomized Approximate Programs*

Tests if the **implementations** satisfies the algorithm’s specifications

The specification provided in a **formal notation**

- Generate inputs according to different distribution
- Gather samples and aggregate data
- Select appropriate statistical test

*Keyur Joshi, Vimuth Fernando, and Sasa Misailovic. 2019. Statistical algorithmic profiling for randomized approximate programs. (ICSE '19)

AxProf : Count-min Sketch Accuracy Testing

Math Specification: $P\,[\,error < N\; * \epsilon] > 1\; - \;\delta$

AxProf specification:

```
Input list of (list of real);
Output list of (list of int);
forall i in unique(Input)
Probability over runs
   [error(i, Input, Output) < |Input| * epsilon] > 1 - delta
```

AxProf: PASS/FAIL

Setting algorithm parameters

How to set number of hash functions (h) and number of bins (w)?

Analytical specification:

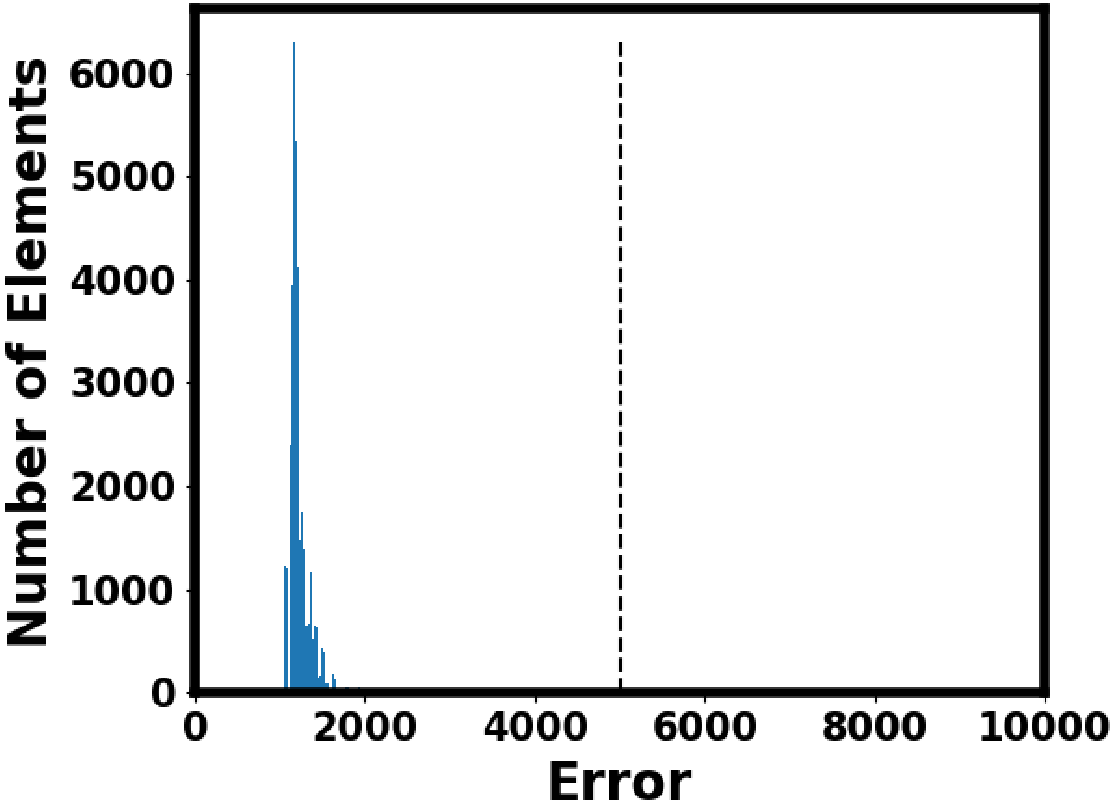
$$P[\text{ error} < N * \epsilon] > 1 - \delta$$

To achieve this guarantee with minimum memory usage

$$w = \lceil 2.718/\epsilon \rceil,\ h = \lceil \ln(1/\delta) \rceil$$

Example:

P[error < 5000] > 0.99 ⇒ w=534, h=5



Observed errors for a randomly generated dataset in an implementation of Count-min

Analytical Error Guarantees Are Conservative

Take into account worst-case scenarios or perform average case analysis for a large input domain

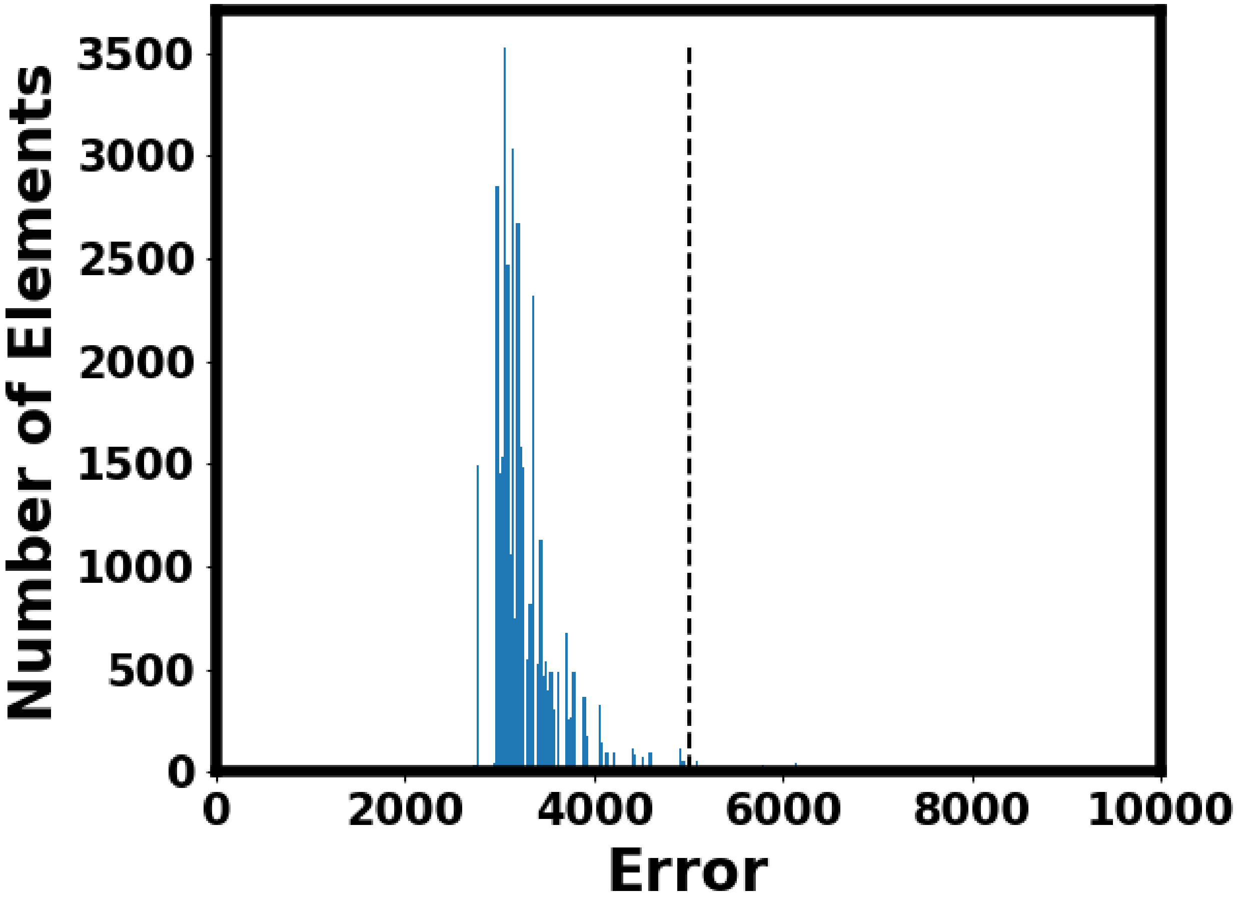
Algorithm implementers can implement different behavior than specified

- Use of polyalgorithms
- Allocate more resources than required (Eg: Bigger arrays)

For some applications it is not possible to derive analytical models due to complex interactions among parameters (e.g., SFFT)

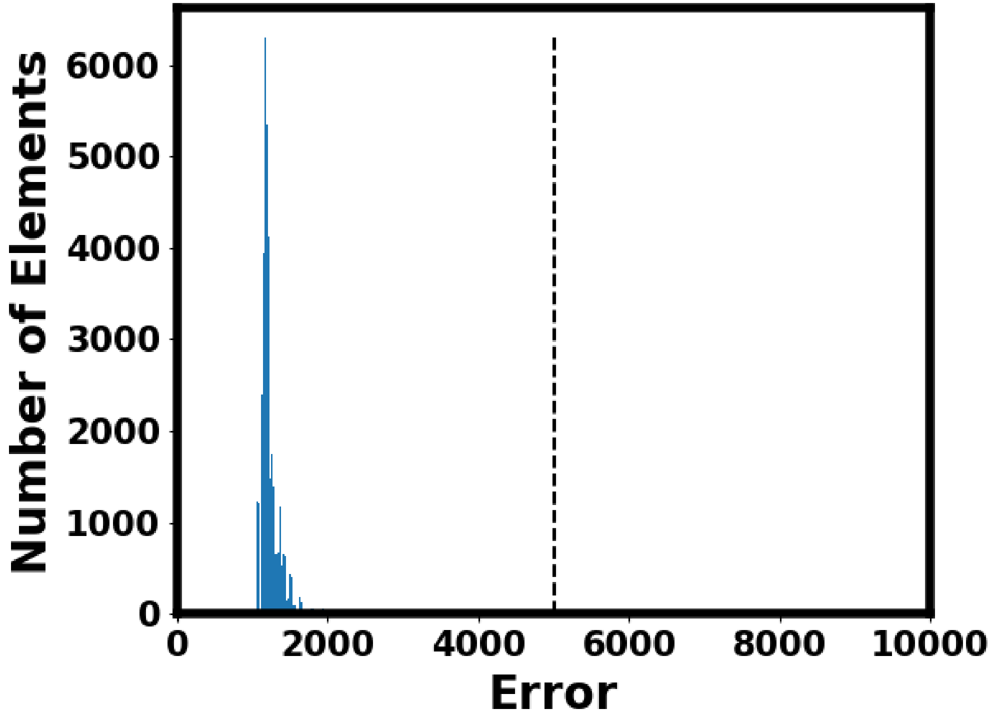
Alternative: Build **empirical models** to identify algorithm parameter values that **satisfy a user's accuracy requirements** while **optimally utilizing resources** **(And satisfying the analytical accuracy guarantee)**

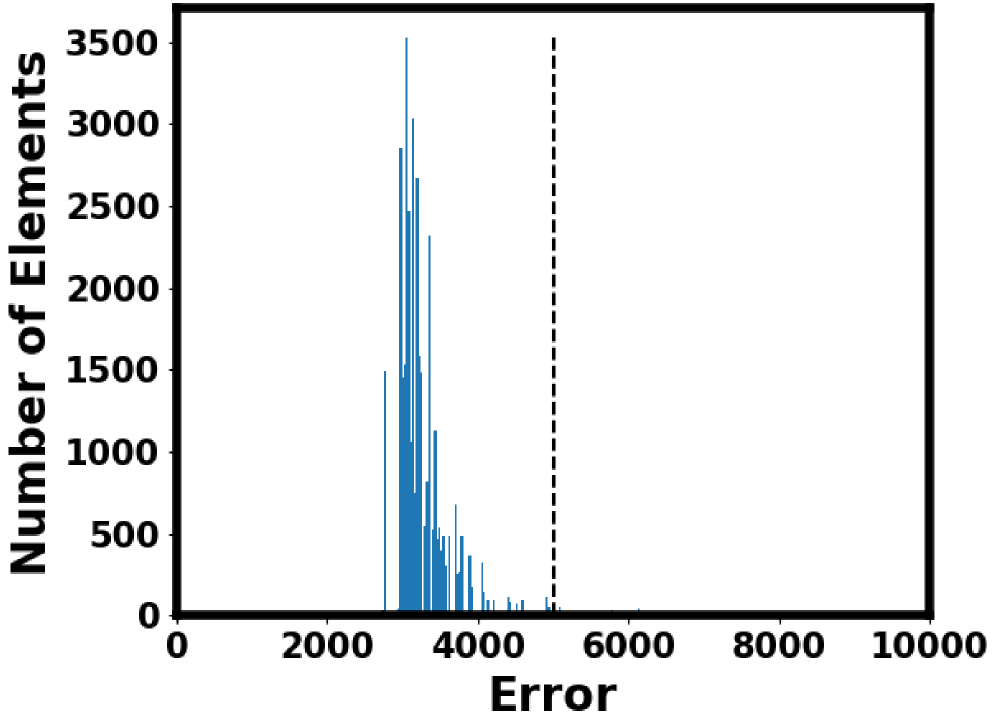
Setting algorithm parameters: Count-min



Observed errors for a randomly generated dataset in an implementation of Count-min

Setting algorithm parameters: Count-min



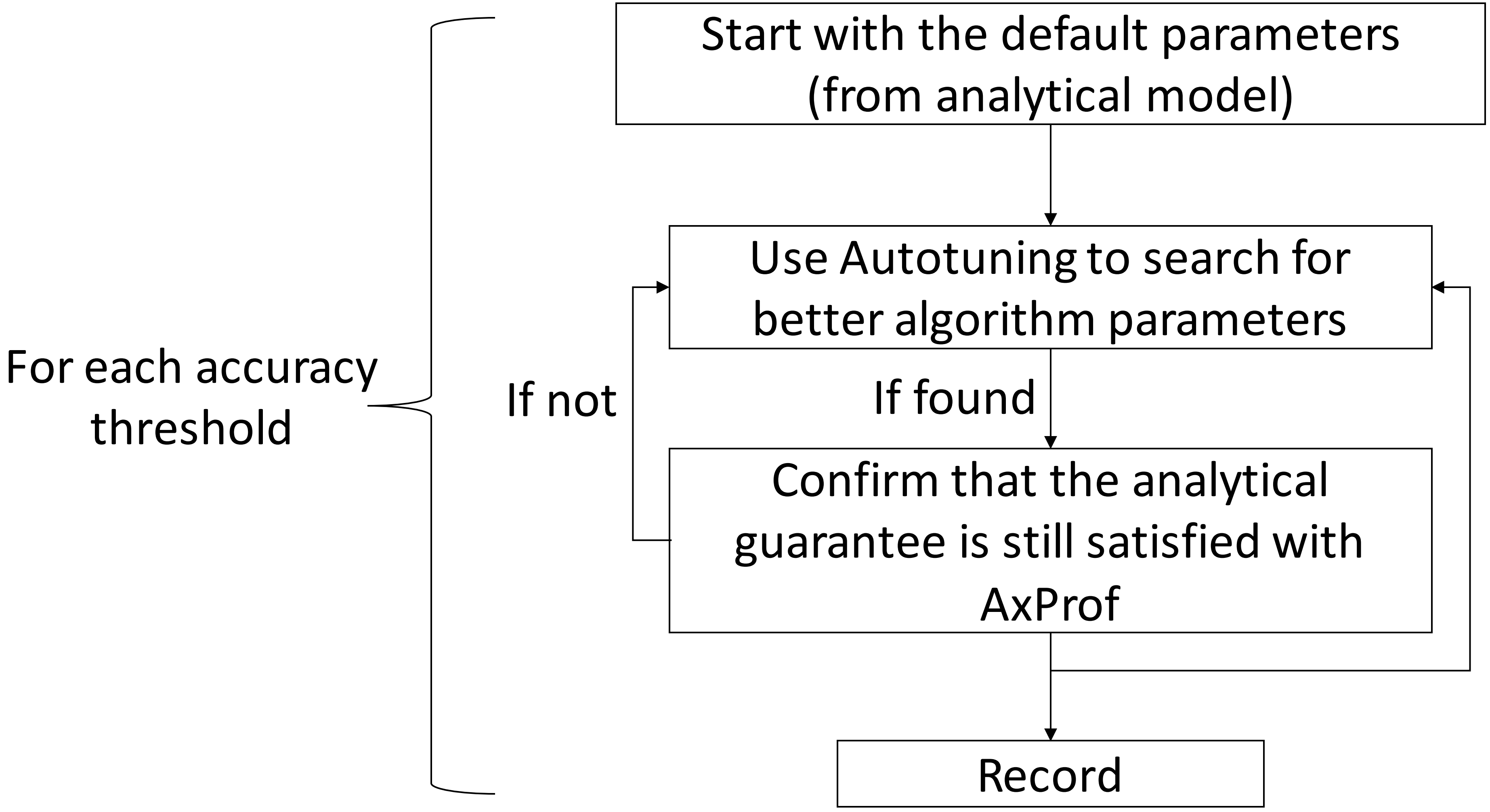


Count-min Sketch Empirical Model of Accuracy

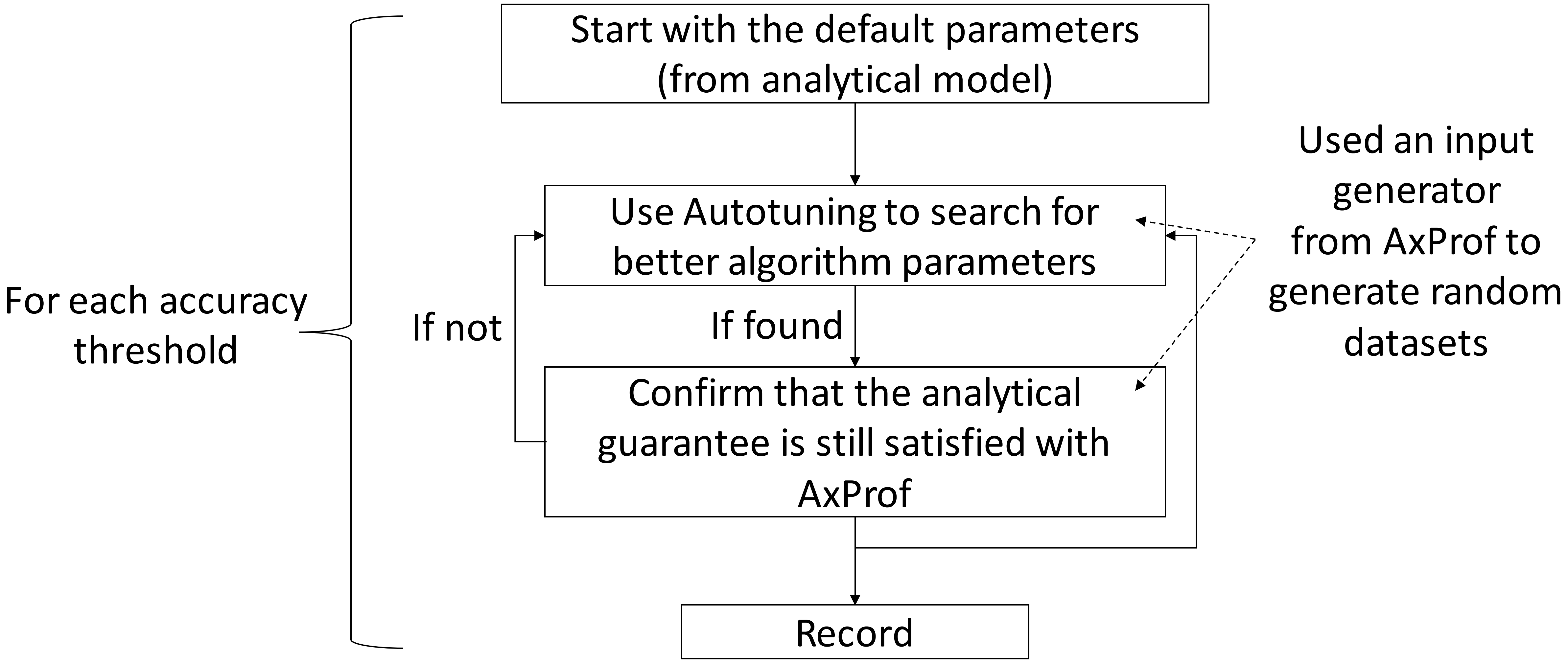
- Identify a representative input set – Lists of integers drawn from Zipf distributions (using AxProf input generator)
- Identify the possible configurations – the ranges of tunable parameters - e.g., w:[1-1000], h:[1-10]
- A tuning objective for each algorithm - optimize memory usage

- We used OpenTuner to identify optimal parameter values for error thresholds **that also satisfy the analytical specification**

Building Empirical Models



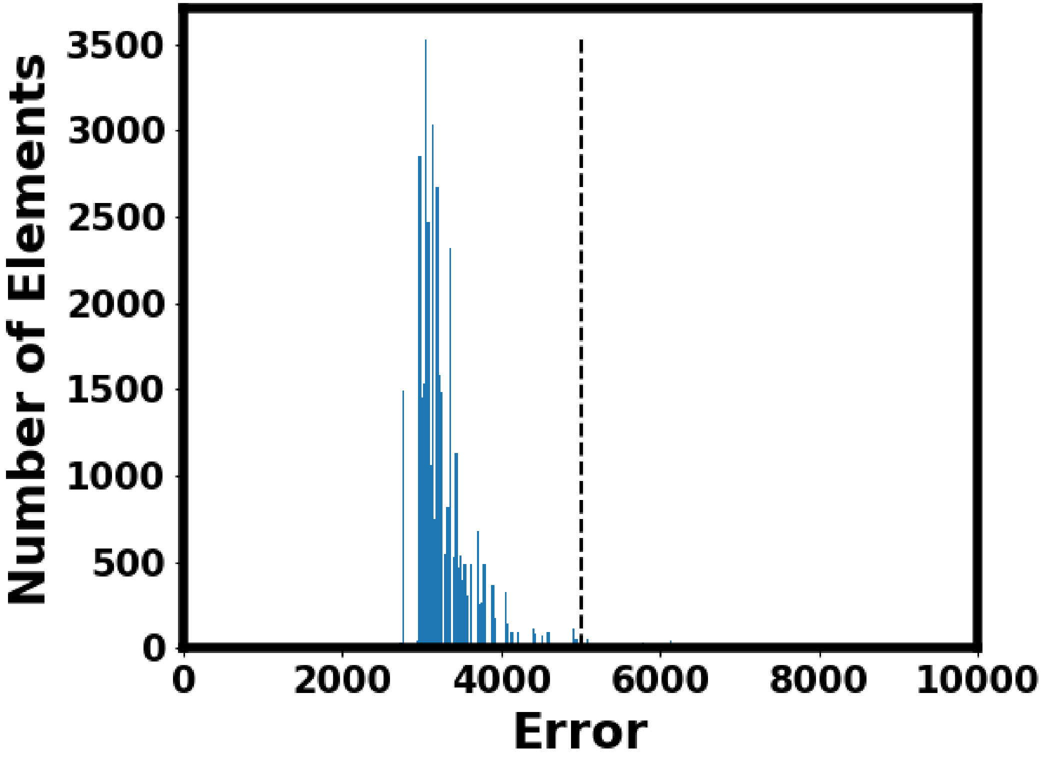
Building Empirical Models



Setting Algorithm Parameters: Count-min

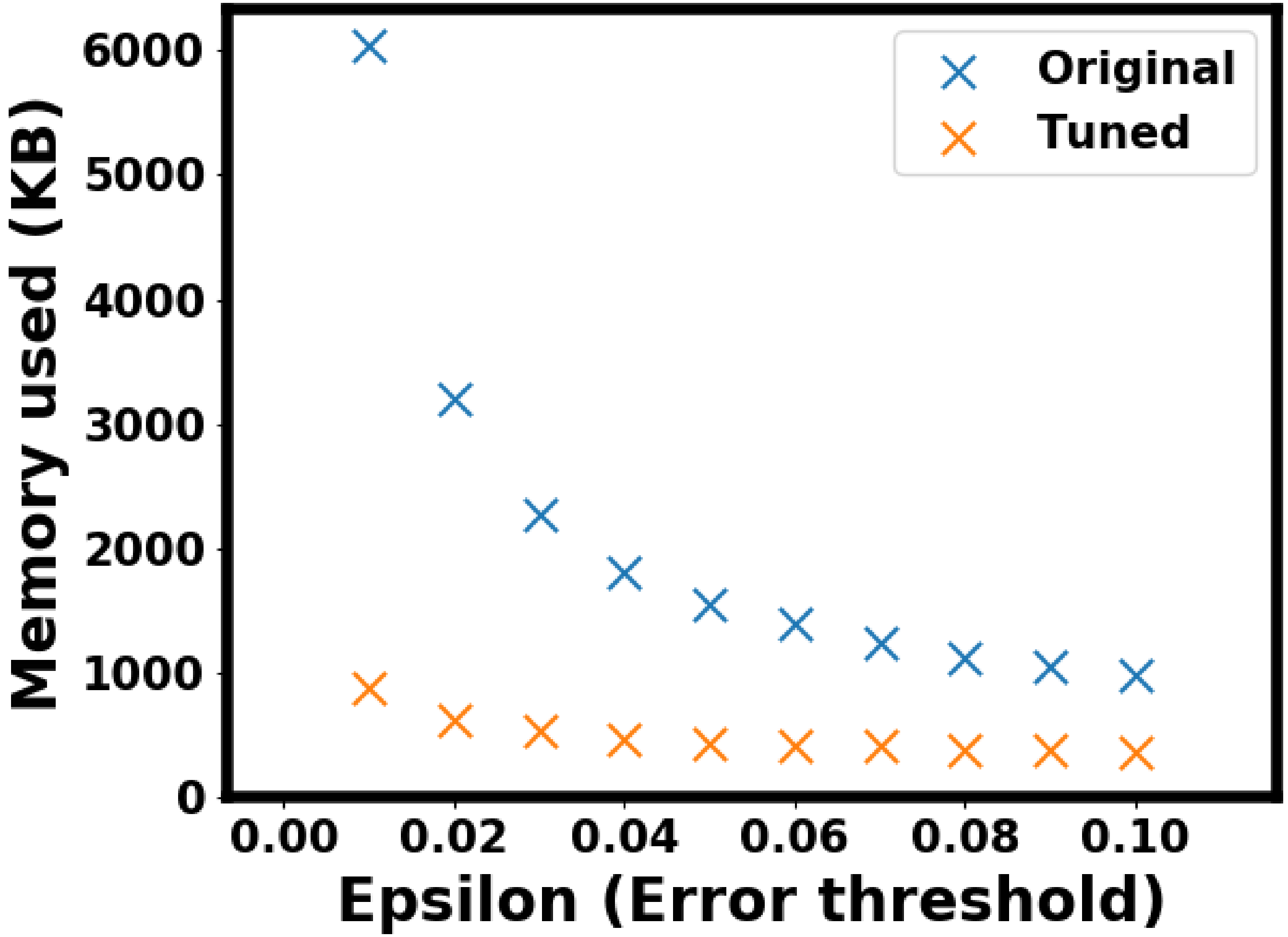
The algorithm finishes 30% faster and uses 50% less memory

(P[error < 5000] > 0.99 ⇒ w=396, h=3)



Observed errors for a randomly generated dataset in an implementation of Count-min

Benefits of an Empirical(tuned) Model



For the specification:

P[error < N * Epsilon] > 1 - δ

Future Directions: Adaptive Algorithms

- When the algorithm accuracy is data dependent - algorithm parameters can be set based on the input to achieve optimal performance
- However, in many of the algorithms the input is very large, therefore pre-analyzing the data may not be possible
- Requires low cost initial analysis of huge data sets

Future Directions: Runtime Monitoring

- An implementation can periodically estimate the error at runtime
- If that estimate starts to exceed an acceptable threshold, issue warnings or change to a more accurate configuration of the algorithm
- Error estimates need to be low cost for benefits

Future Directions: Comparing Implementations

- Two implementations that satisfy the same analytical specifications of the algorithm can have widely varying behavior
- Optimal behavior can be used to compare implementations of the same algorithm

Conclusion

Randomized approximate algorithms have conservative analytical probabilistic accuracy specification

Hybrid models of accuracy/performance can provide better resource usage while providing similar guarantee

Future directions

- Adaptive algorithms
- Runtime monitoring
- Reducing offline training time
- Selecting among multiple implementations