Efficiently Searching In-Memory Sorted Arrays: Revenge of the Interpolation Search?

-Yizhou Mao, Chengjun Wu

Agenda

- What’s the interpolation search?
- How it works?
- Why to use it?
- SIP search & TIP search
- Experiment
- Conclusion

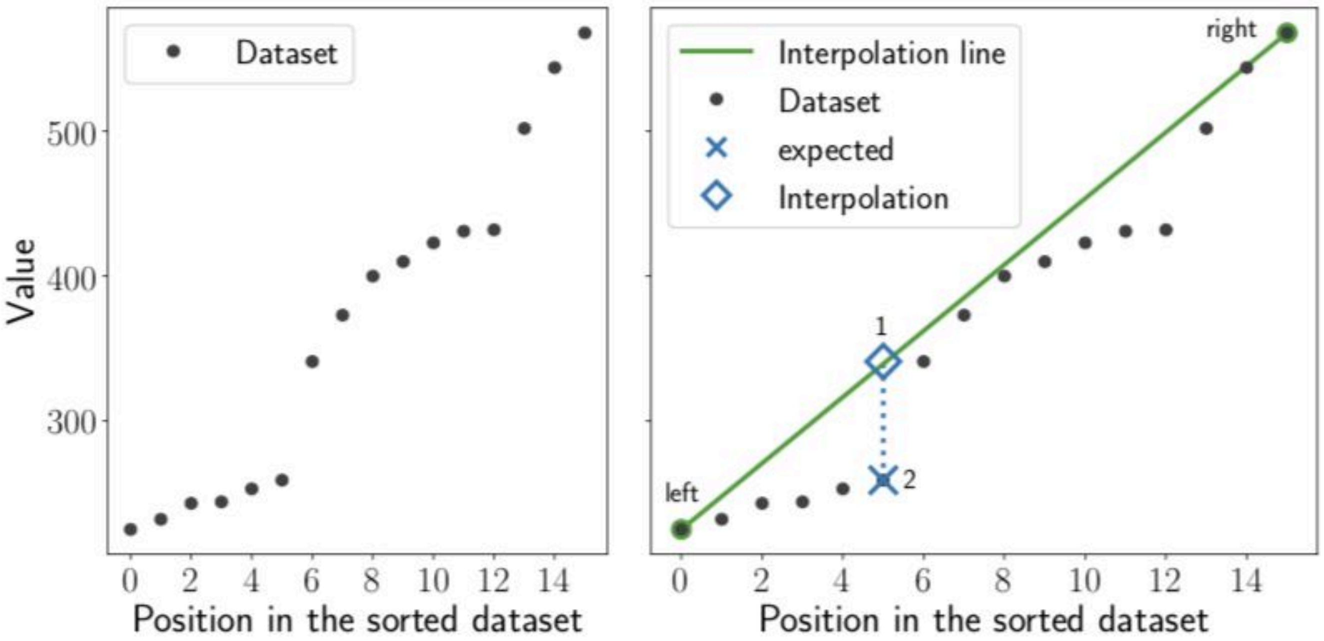
What
How
Why

Interpolation search

What’s the Interpolation Search

- Similar to Binary Search
- Assumes a linear relationship between value and its position
- Do linear interpolation to find the position of targeted value

How does Interpolation Search work



Interpolation Search Math

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

$$x = \frac{(y - y_1)(x_2 - x_1)}{y_2 - y_1} + x_1$$

How does Interpolation Search work – e.g.

Index	0	1	2	3	4	5	6	7	8	9
Value	2	9	30	32	38	47	61	69	79	81

How does Interpolation Search work – e.g.

	●									●
Index	0	1	2	3	4	5	6	7	8	9
Value	2	9	30	32	38	47	61	69	79	81

How does Interpolation Search work – e.g.

Index	0	1	2	3	4	5	6	7	8	9
Value	2	9	30	32	38	47	61	69	79	81

$$x = \frac{(61-2)(9-0)}{81-2} + 0 \approx 6.7215 \approx 7$$

How does Interpolation Search work – e.g.

	●							○		●
Index	0	1	2	3	4	5	6	7	8	9
Value	2	9	30	32	38	47	61	69	79	81

61 < 69

How does Interpolation Search work – e.g.

	●							●		
Index	0	1	2	3	4	5	6	7	8	9
Value	2	9	30	32	38	47	61	69	79	81

61 < 69

How does Interpolation Search work - e.g.

Index	0	1	2	3	4	5	6	7	8	9
Value	2	9	30	32	38	47	61	69	79	81

$$x = \frac{(61-2)(6-0)}{61-2} + 0 = 6$$

How does Interpolation Search work – e.g.

Index	0	1	2	3	4	5	6	7	8	9
Value	2	9	30	32	38	47	61	69	79	81

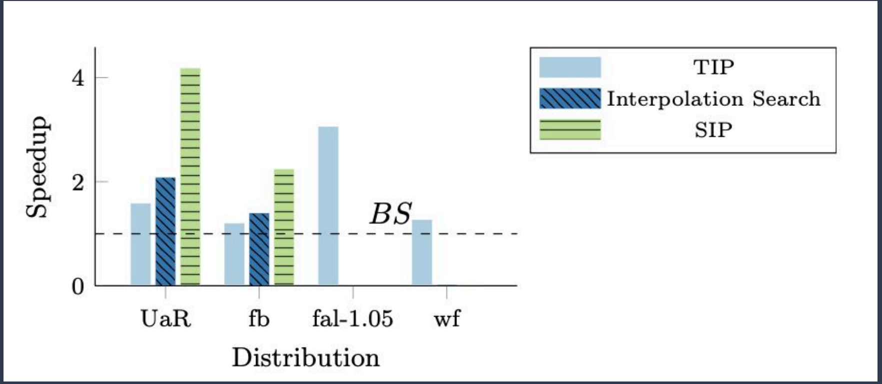
61 == 61

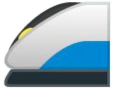
$$x = \frac{(61-2)(6-0)}{61-2} + 0 = 6$$

Why to use Interpolation Search?

- Hardware trend favors more in-memory computation
- Fewer comparison needed, Fewer cache misses
- Additional computation needed, but cost less than memory access

SIP Search & TIP Search



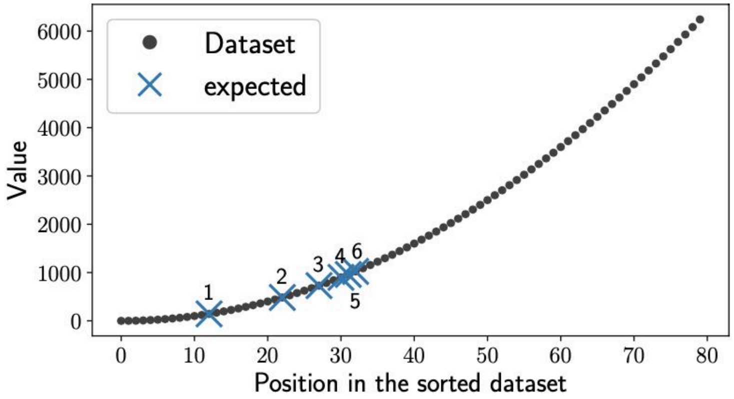
	Uniform	Skewed
SIP		
TIP		
Regular		

SIP Search

- Efficient for **uniform** distributed data
- Interpolation Search with applying optimizations
  - ❑ Guard
  - ❑ Slope Reuse
  - ❑ Fixed-point arithmetic

Optimization: Guard

Problem: Marginal Benefits of interpolation diminish as iteration go through



Optimization: Guard

Problem: Marginal Benefits of interpolation diminish as iteration go through

- Switch to **Sequential Search** when getting close to target value
- **Loop unrolling** & **sentinels**

Optimization: Guard

- Loop unrolling

```
for (i = 1; i <= 60; i++)
    a[i] = a[i] * b + c;
```

→

```
for (i = 1; i <= 58; i+=3){
    a[i] = a[i] * b + c;
    a[i+1] = a[i+1] * b + c;
    a[i+2] = a[i+2] * b + c;
}
```

Optimization: Guard

- Sentinels

```
(data, key){
    int index = 0;
    while(index < data.length)
        if(data[index++] == key)
          return index - 1;
    return -1;
};
```

```
(data,key){
    int index = 0;
    data.add(key); //add sential to start or end
    while(data[index++] != key){}
    data.remove(key); //remove sential
    return (--index == data.length) ? -1 : index;
 };
```

Optimization: Slope Reuse

Problem: Computing Slope for each iteration is costly

- **Reuse slope** calculated in the **first** interpolation

Optimization: Fixed-point arithmetic

Problem: Multiplication by the slope of interpolation line accounts for major arithmetic cost in interpolation search.

- Bypass integer division
- Turn to **fixed-point arithmetic**

Optimization: Fixed-point arithmetic

$$expected = left + (y^* - V[left]) \frac{right - left}{V[right] - V[left]} \tag{1}$$

Optimization: Fixed-point arithmetic

$$
\begin{aligned}
y * s = y^* \frac{p}{q} &\approx \lfloor y^* \lceil \frac{2^{64} p}{q} \rceil \div 2^{64} \rfloor \\
&= \lfloor y * p' \div 2^{64} \rfloor, p' = \lceil \frac{2^{64} p}{q} \rceil && (3) \\
&= y \otimes s' && (4)
\end{aligned}
$$

Optimization: Fixed-point arithmetic

E.g: $29 / 2^2 = 7$.

- $29 = 2^4 + 2^3 + 2^2 + 2^0 = 11101$
- $11101 >> 2 = 111 = 7$

TIP Search

- Flexible for **non-uniform** distributed data
- Interpolation Search with applying optimizations
  - ❑ Guard
  - ❑ 3-Point Interpolation

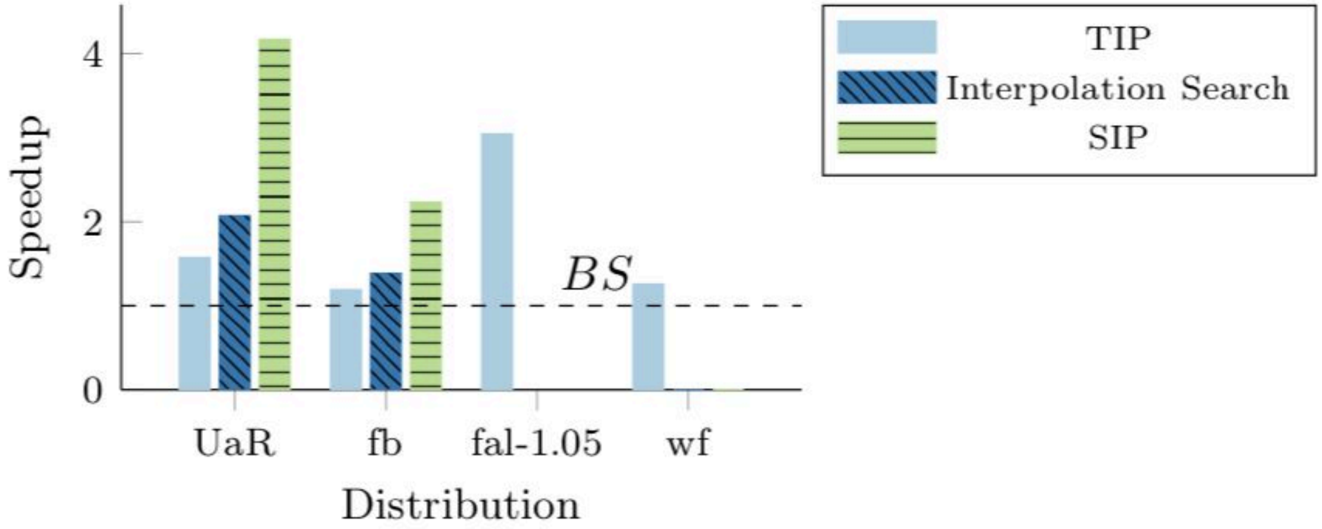
Optimization: 3-Point Interpolation

Problem: How to fit non-linear distributed data through interpolation method?

- Adopt **3-Point interpolation** rather than 2 endpoints interpolation

Experiment

Comparison to Binary Search



Searching on Uniform Dataset

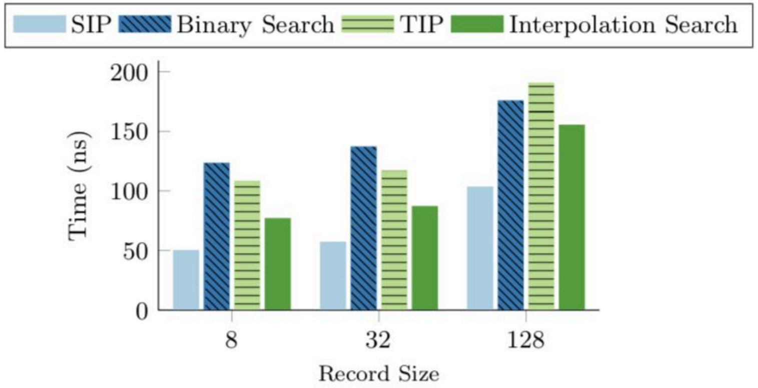


Figure 5: Comparison of SIP, TIP and Binary Search on the *fb_ids* dataset, over different record sizes.

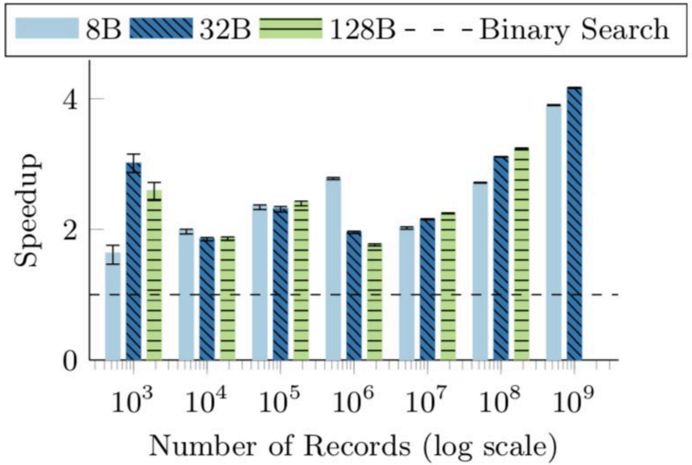


Figure 6: Speedup of SIP compared to Binary Search for different dataset and record sizes, on the *UaR* dataset. 10^9 **records of size 128B exceed the memory capacity.**

Searching on Uniform Dataset

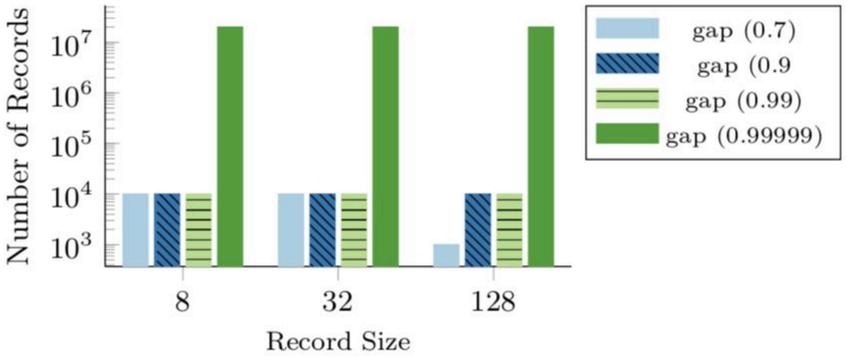


Figure 8: The dataset size (Number of Records) where SIP becomes faster than Interpolation-Sequential. For smaller sizes, Interpolation-Sequential is faster, for larger sizes, SIP is faster.

Searching on Non-Uniform Dataset

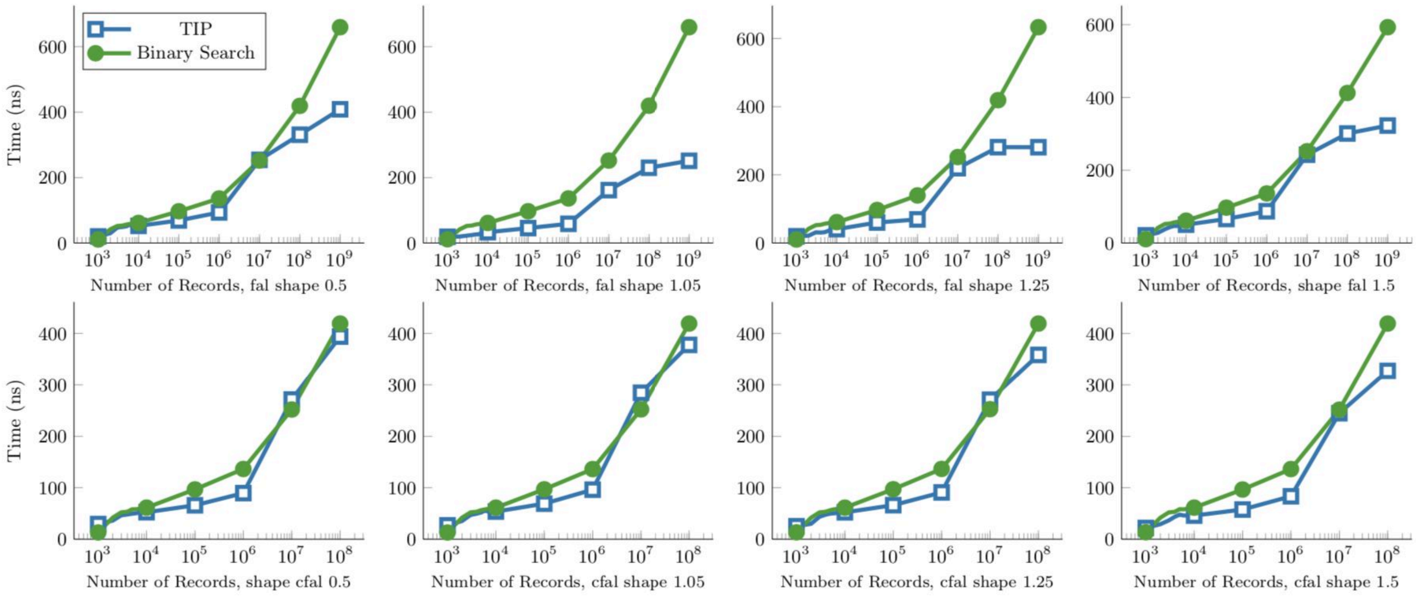


Figure 9: Time to perform a search for TIP and Binary Search for different dataset sizes on synthetic, non-uniform datasets. Record size is 8 Byte records, the results for other record sizes are similar and we omit them (X axis is in log scale).

Searching on Non-Uniform Dataset

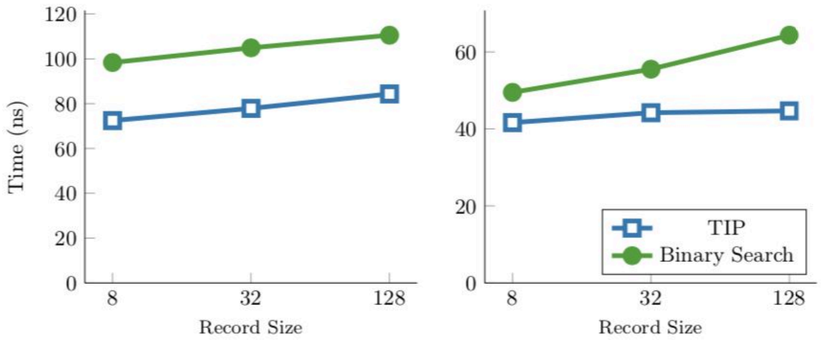


Figure 11: TIP compared to Binary Search on the $freq1$ (left) and $freq2$ (right) datasets for different record sizes.

Conclusion

Conclusion

- Go over naive interpolation search
- Research several optimization techniques
- Performance comparison to Binary Search