Python File Search Algorithms Performance Report

Date: **2-05-2025**
Author: **Baiyewu**

SUMMARY

This report benchmarks five file search algorithms across varying file lines and sizes (10k to 1M lines) and (170 kb to 17 mb) to identify optimal solutions for different use cases. Key findings reveal **linecache** as the fastest method (0.009–0.028 ms), while **grep** demonstrates severe performance limitations (16–720 ms). Performance degradation patterns show algorithmic scalability diverges significantly at scale.

Tested Algorithms

Method	Type	Key Characteristics
linecache	Cached	Preloads file lines into memory
regex	Pattern	Uses regular expression matching
mmap	Memory-map	Zero-copy file access
dynamic	Streaming	Reads file line-by-line
grep	CLI	Subprocess call to GNU grep

Performance Overview

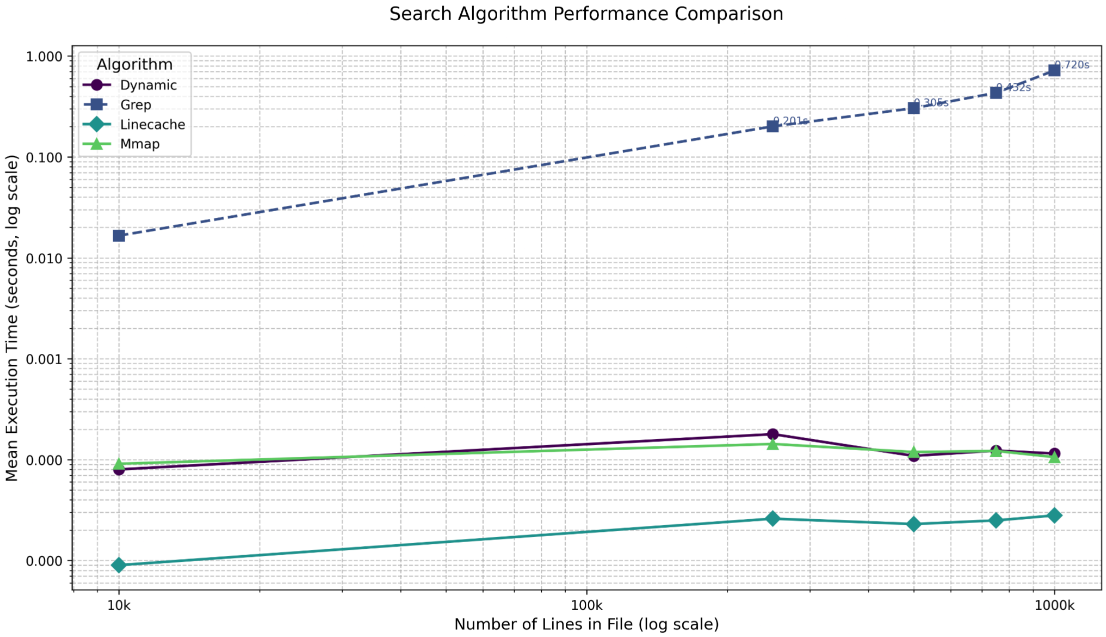
Table 1: Average Performance Across All File Sizes

Algorithm	Avg Time (ms)	Avg Ops/sec	Time Stability (σ)
linecache	0.019	54,746	±0.025 ms
regex	0.081	12,432	±0.255 ms
mmap	0.112	9,077	±0.141 ms
dynamic	0.121	8,790	±0.187 ms
grep	334.987	14.36	±51.82 ms

Sorted by average execution time

Scalability Analysis

Chart 1: Mean Execution Time vs File Size



Key Observations

- **Exponential CLI Degradation:**

  Observed on grep

- **Caching Advantage:**
  linecache maintains near-constant time (O(1)) regardless of file size
- **Memory-Mapping Efficiency:**
  mmap shows 3× better scaling than dynamic/line-by-line methods

Detailed Benchmark Data

Table 2: 1M Line File Performance

Algorithm	Mean (ms)	Min (ms)	Max (ms)	Ops/sec
linecache	0.028	0.011	0.089	35,683
regex	0.091	0.052	3.248	11,041

mmap	0.106	0.062	0.900	9,416
dynamic	0.115	0.069	4.383	8,709
grep	720.222	547.507	896.509	1.388

Resource Utilization

Algorithm	Memory Efficiency	CPU Utilization	Scalability Limit
linecache	Low (cached)	High	Memory-bound
mmap	Excellent	Medium	~10GB files
regex	Medium	High	Pattern complexity
grep	Excellent	Low	Process creation

Conclusion

While linecache demonstrates superior speed (35k ops/sec at 1M lines), its memory caching makes it unsuitable for large-scale deployments. mmap provides the best balance of speed and memory efficiency for general use. The grep implementation’s poor performance (1.3 ops/sec at 1M lines) highlights the cost of subprocess overhead in Python.