Outline of EE

Seungjun Lee

February 22, 2026

Abstract

Title: When Asymptotic Complexity Fails: An Empirical and Cost-Based Study of Binary and Fibonacci Heaps in Python

RQ: Why does the theoretical asymptotic advantage of Fibonacci heap in Dijkstra's algorithm not translate into practical runtime improvements in Python implementations?

Category: Computer Science

Contents

1 Introduction

Context

- Dijkstra's algorithm widely used for shortest path problems
- Priority queue choice affects theoretical time complexity
- Pure Dijkstra: $O(E + V^2)$
- Binary heap: $O(E \log V)$
- Fibonacci heap: $O(E + V \log V)$
- Theoretical advantage not clearly observed in practical implementations

Outline of argument

- Graph structure influences decrease-key frequency
- Decrease-key frequency determines theoretical advantage
- Empirical runtime does not reflect the theory
- Operation level cost explains the gap between the thoery and reality

Scope

- Python implementation only
- Synthetic directed graphs (using outdegree-based generation)
- Lognormal edge weight distribution
- Nodes: 2000–16000, density: 0.00015–0.0384, stds: 1000–16000, trials: 200
- Single-source single-target shortest path

Worthiness

- Evaluates limits of asymptotic complexity in practice
- Provides empirical basis for heap selection
- Bridges theory and implementation level cost analysis

2 Theoretical Framework

- Formal runtime models:
  - $T_B = c_1 E \log V$
  - $T_F = c_2 E + c_3 V \log V$
- Role of decrease-key in Dijkstra
- Relation between relax_success and decrease-key calls
- Theoretical crossover condition:

$$T_F < T_B$$

3 Methodology

Development

- Real-world road dataset used to estimate distribution of real-world data
- Lognormal parameters derived from real data
- Synthetic graph generation with various nodes, density, std
- Measurement metrics:
  - reached
  - extract_min_calls
  - relax_attempts
  - relax_success
  - relax_success_ratio
  - runtime

Evidence

- Experimental configuration

```
python -V
Python 3.12.12

pip -V
```

```
pip 26.0.1

python -c "import platform; print(platform.platform())"
Linux-6.8.12-17-pve-x86_64-with-glibc2.41

lscpu | head
Architecture:                x86_64
CPU op-mode(s):              32-bit, 64-bit
Address sizes:               48 bits physical, 48 bits virtual
Byte Order:                  Little Endian
CPU(s):                      16
On-line CPU(s) list:         1-3,8
Off-line CPU(s) list:        0,4-7,9-15
Vendor ID:                   AuthenticAMD
Model name:                  AMD Ryzen 7 7700 8-Core Processor
CPU family:                  25
```

Analysis

- Justification of parameter intervals
- Retain stability by 200 trials

Connection

- Method is the fundament of runtime investigation

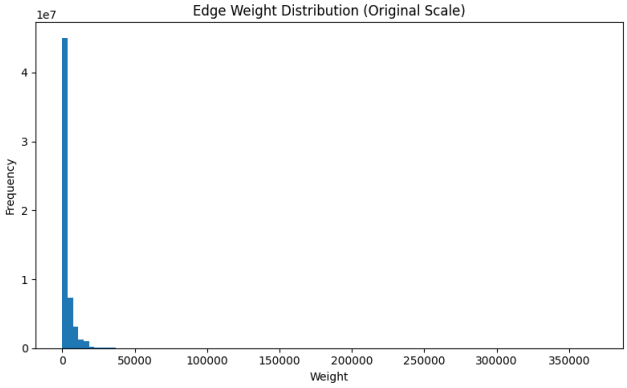
4 Real Data Analysis

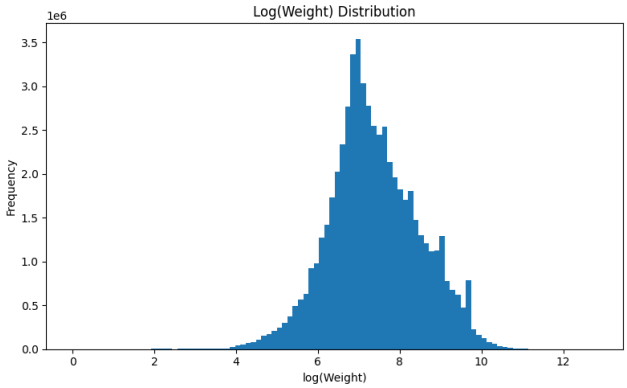
Development

- Get real data from DIMACS
- Extraction of edge weight distribution
- Calculation of mean and std

Evidence

- Edge weight distribition





- Summary statistics tableofcontents

Key	Value
Total edges	58333344
Mean	2950.322
Std	4071.694
Min	1
Max	368855

Analysis

- Lognormal approximation calculation
- Justification of synthetic distribution parameters

Connection

- Synthetic data is based on real-world data

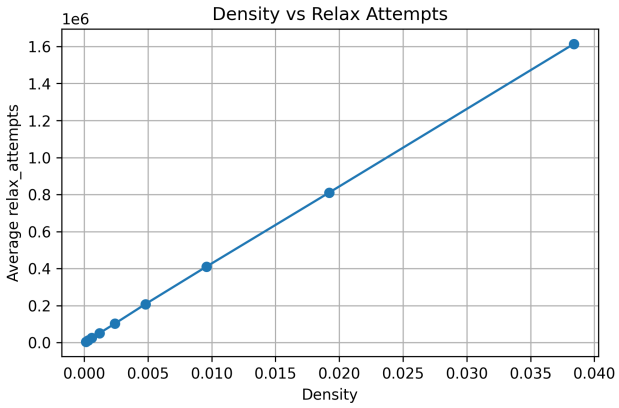
5 Relationship between variables

Development

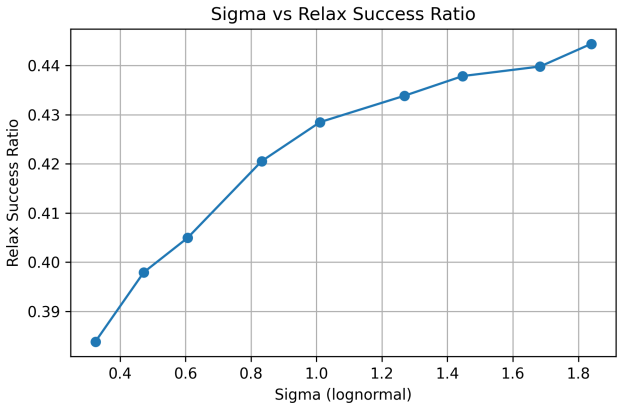
- Density vs relax_attempts
- Sigma vs relax_success_ratio
- Decrease-key vs speedup

Evidence

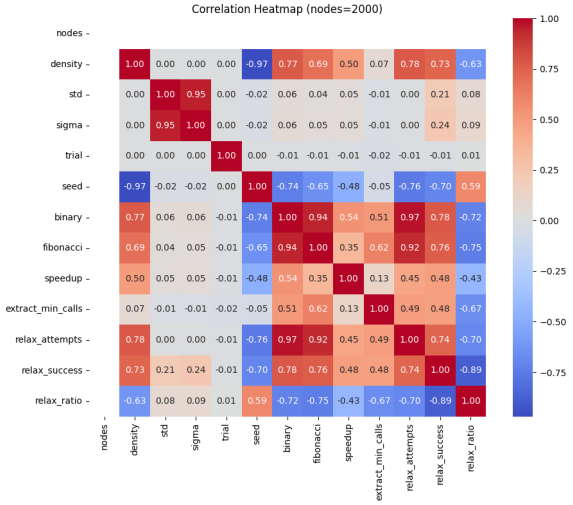
- Density vs relax_attempts plot



- Sigma vs relax_success_ratio platform



- Decrease-key vs speedup correlation



Analysis

- Density increases relax attempts nearly linearly
- Higher variance increases relax_success ratio
- Speedup weakly correlated with decrease-key frequency

Connection

- Structural conditions necessary but insufficient for crossover

6 Runtime

Development

- Multiple linear regression of runtime
- Operation level cost estimation

Evidence

- Regression summary table

algorithm	intercept	$coef_{add}$	$coef_{extract}$	$coef_{relax}$	$coef_{decrease}$
binary	0.000492	-3.496595e-07	3.259253e-07	9.095429e-08	1.702164e-07
fibonacci	-0.001191	7.446706e-07	3.938917e-07	9.333419e-08	1.504052e-06

- R^2 comparison between models

algorithm	r_2	$n_{samples}$
binary	0.980930	54801
fibonacci	0.980877	54801

Analysis

- Fibonacci decrease-key cost significantly larger in Python
- Constant factors dominate asymptotic differences

Connection

- Explains absence of empirical speedup

7 Deriving the Crossover Condition

Development

- Substitute empirical coefficients into:

$$T_F < T_B$$

Evidence

- Derived inequality expression

Analysis

- Required conditions exceed practical graph scale
- Python constant overhead prevents crossover

Connection

- Theoretical superiority does not always linked to practical improvement

8 Conclusion

Conclusion

- Fibonacci heap does not outperform Binary heap in Python
- Operation level costs dominate asymptotic complexity
- Crossover condition impractical under tested environment

Limitation & Evaulation

- Python-only implementation
- Synthetic graph model assumptions
- No low-level memory/cache analysis

Summarize main points

- Graph structure affects decrease-key frequency
- Empirical runtime does not follow asymptotic expectation
- Constant factors determine real-world performance