

Predicting Dijkstra Runtime from Graph Properties

An Operation-Decomposed Model with Binary Heap

SEUNGJUN LEE

April 14, 2026

Contents

1	Introduction	3
2	Background	3
2.1	Dijkstra’s Algorithm	3
2.1.1	Time Complexity	3
2.2	Priority Queue	3
2.2.1	Time Complexity	4
2.3	Runtime Model	4
3	Methodology	4
3.1	Research Hypotheses	4
3.2	Experimental Environment	5
3.3	Real Data	5
3.4	Synthetic Graph Generation	5
3.5	Analysis Strategy	6
3.5.1	Decrease-Key Call Count (α) Modelling	6
3.5.2	Operation Unit Cost Modelling	6
3.5.3	Integrated Prediction	6
3.6	Validation Metrics	7
4	Results	7
4.1	Real Data Properties	7
4.2	Decrease-Key Call Number Prediction	7
4.3	Operation Unit Cost Analysis	9
4.4	Integrated Runtime Prediction	10
5	Discussion	12
5.1	Phase Transition as Regime Boundary	12
5.2	Cache Effects in Unit Cost	12
5.3	Asymptotic vs Empirical Cost	12
5.4	Limitations	13
6	Conclusion	13
6.1	Summary of Contributions	13
6.2	Practical Implications	13

6.3	Future Work	14
-----	-----------------------	----

1 Introduction

Shortest-path computation is fundamental to a broad class of real-world applications. Autonomous navigation in robotic systems relies on optimal path planning as a core primitive (Waga et al., 2025), and intelligent transportation systems require fast, accurate routing for in-vehicle guidance and automated vehicle dispatch (Fu et al., 2006). As graph sizes grow, the tension between solution quality and computation time becomes critical: exact algorithms guarantee optimality but scale poorly, while heuristic methods such as A* (Hart et al., 1968) sacrifice optimality for speed.

Dijkstra’s algorithm (Dijkstra, 1959) remains the canonical exact method for single-source shortest paths on non-negative-weight graphs. Despite its theoretical guarantees, practical deployment is hindered by the difficulty of predicting its runtime before execution. When graphs are large, memory consumption and processing time can be prohibitive (Fuhao & Jiping, 2009), yet practitioners must decide in advance whether Dijkstra is feasible for a given instance. Heuristic approaches such as A* are therefore preferred in many transportation applications (Fu et al., 2006), even when the optimality guarantee of Dijkstra would be desirable.

This asymmetry motivates a different question: rather than replacing Dijkstra, can its runtime be predicted from observable graph properties alone, enabling informed pre-execution decisions? If so, practitioners could determine whether Dijkstra is viable for a specific instance before committing computational resources.

This study addresses the following research question:

Can the wall-clock runtime of binary-heap Dijkstra be predicted using only N (node count), d (edge density), and σ (log-scale standard deviation of edge weights)?

The proposed model decomposes runtime into distinct operation categories, models each component as a function of graph properties or N , and combines them into a closed-form prediction formula. A critical structural discovery is a phase transition at average degree $\bar{k} = 1$ that defines two fundamentally different operating regimes.

2 Background

2.1 Dijkstra’s Algorithm

Dijkstra’s algorithm (Dijkstra, 1959) solves the single-source shortest-path problem on a directed weighted graph $G = (V, E)$ with non-negative edge weights. Starting from a source vertex, it maintains a priority queue of tentative distances and iteratively extracts the minimum-distance vertex, relaxing all outgoing edges. An edge relaxation succeeds — triggering an update — when a shorter path to the neighbor is discovered.

2.1.1 Time Complexity

In its original array-based form, the algorithm requires $O(V^2)$ time. The bottleneck is the repeated minimum extraction, which costs $O(V)$ per iteration over V iterations (Dijkstra, 1959).

2.2 Priority Queue

Replacing the linear scan with a priority queue reduces the cost of minimum extraction. Two well-studied options are the binary heap (Williams, 1964) and the Fibonacci heap. Although Fibonacci heaps achieve

superior asymptotic complexity for decrease-key ($O(1)$ amortised versus $O(\log V)$), empirical benchmarks consistently show that binary heaps outperform Fibonacci heaps in practice due to lower constant factors and better cache behaviour (Idowu et al., 2025). This study therefore uses a binary heap implementation.

2.2.1 Time Complexity

With a binary heap, the standard complexity analysis yields $O((V + E) \log V)$. More precisely: each of the V vertices is added once ($O(\log V)$ per add), extracted once ($O(\log V)$ per extraction), and each of the E edges is relaxed. Each successful relaxation triggers a decrease-key at $O(\log V)$. Assuming every relaxation succeeds and counting V decrease-keys yields $O((V + E) \log V)$.

2.3 Runtime Model

The standard $O((V + E) \log V)$ bound conflates four operationally distinct contributions, omits the fraction of relax attempts that actually succeed, and absorbs all constants. This study proposes the following additive decomposition:

$$\text{Runtime} = V \cdot \text{UC}_1 + E \cdot \text{UC}_2 + \alpha \cdot \text{UC}_3 \quad (1)$$

where:

- $\text{UC}_1 = (k_1 + k_2) \log V$: combined unit cost of one add and one extract-min (each called exactly V times);
- $\text{UC}_2 = k_3$: unit cost per relax-attempt iteration (called exactly E times);
- $\text{UC}_3 = k_4 \log V$: unit cost per decrease-key call;
- $\alpha = E \times \text{RSR}$: total number of decrease-key calls, where $\text{RSR} \in [0, 1]$ is the relax-success ratio.

The quantities RSR and the three unit costs are unknowns to be estimated from data. Unlike the asymptotic bound, this model explicitly accounts for the fact that not every relax attempt triggers a decrease-key, and treats per-operation costs as functions of N to capture empirical hardware effects.

3 Methodology

3.1 Research Hypotheses

Three hierarchical hypotheses guide the analysis:

H1 The relax-success ratio RSR is a deterministic function of (N, d, σ) .

H2 Operation unit costs are functions of N alone.

H3 Combining the RSR model and the unit-cost models yields accurate runtime predictions.

3.2 Experimental Environment

All experiments were conducted on a Debian virtual machine with four CPU cores and 8 GB RAM. Dijkstra’s algorithm and the binary heap were implemented from scratch in Python 3.12 to avoid library-level optimisations that would obscure per-operation costs. Wall-clock time was measured for the complete execution of each Dijkstra call.

3.3 Real Data

The DIMACS USA road-distance benchmark (Singh et al., 2025) provides eleven road-network graphs ranging from 264,346 nodes (New York) to 23,947,347 nodes (full USA). These graphs served as a validation target representing real-world conditions.

Edge-weight distributions were characterised for each graph. As shown in Figure 1, the log-transformed weights closely follow a normal distribution, confirming lognormal structure (Singh et al., 2025). The Q–Q plot in Figure 2 corroborates this finding. Summary statistics across graphs yield a typical log-scale standard deviation $\sigma \approx 0.75\text{--}1.12$ and density $d \approx 10^{-7}\text{--}10^{-5}$, parameters that fall within the synthetic training range.

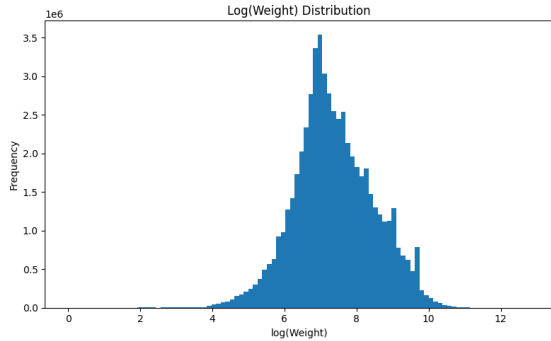


Figure 1: Histogram of log-transformed edge weights for a representative DIMACS graph (BAY). The distribution closely follows a normal curve, confirming lognormal structure.

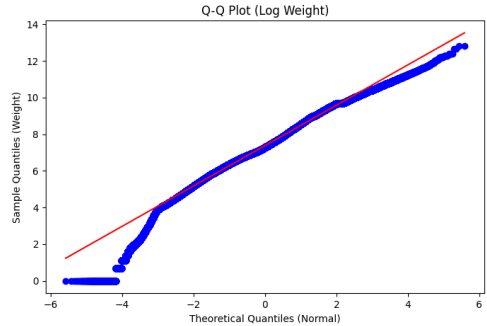


Figure 2: Q–Q plot of log-transformed weights against the normal quantiles. Points align tightly with the diagonal, further supporting lognormality.

3.4 Synthetic Graph Generation

Synthetic Erdős–Rényi random graphs were generated with the following parameter grid:

- $N \in \{2000, 4000, 6000, \dots, 20000\}$ (10 levels);
- $d \in \{10^{-7}, 3 \times 10^{-7}, 10^{-6}, \dots, 0.03\}$ (12 levels, logarithmically spaced);
- $\sigma \in \{0.32, 0.47, 0.61, 0.83, 1.01, 1.27, 1.45, 1.68, 1.84\}$ (9 levels, corresponding to raw-scale standard deviations $\{1000, 1700, 2000, 3000, 4000, 6000, 8000, 12000, 16000\}$ at mean 3000).

Edge weights were drawn from a lognormal distribution with the specified σ and mean 3000. Each (N, d, σ) combination was tested with 400 independent trials, yielding a total of 864,000 measurements. The parameter ranges were chosen to bracket the DIMACS graphs and cover both sparse and moderately dense regimes. Graphs with fewer than two connected components were retained to ensure Dijkstra visits a substantial fraction of vertices.

3.5 Analysis Strategy

3.5.1 Decrease-Key Call Count (α) Modelling

The decrease-key call count α is decomposed as $\alpha = E \times \text{RSR}$. Since $E = d \cdot N(N-1)$ is exactly determined by d and N , the modelling task reduces to estimating RSR.

Plotting RSR against average degree $\bar{k} = (N-1) \cdot d$ reveals a sharp phase transition at $\bar{k} = 1$ (Figure 3): for $\bar{k} \ll 1$ (subcritical regime), graphs are nearly trees or disconnected, so almost every relaxation succeeds and $\text{RSR} \approx 1$; for $\bar{k} \gg 1$ (supercritical regime), the graph is well-connected and RSR decays with increasing density and edge-weight spread.

This transition corresponds directly to the Erdős–Rényi percolation threshold (Erdős & Rényi, 1960), where a giant connected component emerges at $\bar{k} = 1$.

In the supercritical regime ($\bar{k} \geq 1$), controlled experiments confirm that σ and $\bar{k}$ act independently on RSR. Their effects are combined multiplicatively:

$$\text{RSR} = (\sigma^2)^a \cdot (b \ln \bar{k} + c), \quad \bar{k} \geq 1 \quad (2)$$

Nonlinear least-squares regression on the supercritical subset yields $R^2 = 0.9947$.

3.5.2 Operation Unit Cost Modelling

Fitting a single regression of Runtime on V , E , and α across all N causes severe multicollinearity ($\text{VIF} > 10$) because V , E , and α are all functions of N . To circumvent this, a separate linear regression is fitted for each distinct N :

$$\text{Runtime}(N) = \beta_0(N) + E \cdot k_3(N) + \alpha \cdot k_4(N) \quad (3)$$

This extracts per- N estimates of k_3 and k_4 , verifying $\text{VIF} < 3.5$ within each stratum.

The extracted coefficients are then regressed against N :

- $k_3(N)$: the relax-attempt cost decreases with N following an exponential-decay model $k_3 = a_3 e^{-N/N_0} + c_3$ ($R^2 = 0.953$), attributed to CPU cache effects at small N .
- $k_4(N)$: the decrease-key cost grows with heap height as $k_4 \log V$, confirming the theoretical $O(\log V)$ scaling ($R^2 = 0.951$).
- $\beta_0(N)$ (intercept capturing $V \cdot \text{UC}_1$): noisy due to collinearity with V ; a per- N median is used as a constant.

3.5.3 Integrated Prediction

The final predictor combines Equations (1) and (2):

$$\hat{t} = V \cdot \hat{\text{UC}}_1 + E \cdot \hat{k}_3(N) + \hat{\alpha} \cdot \hat{k}_4(N) \cdot \log V \quad (4)$$

where $\hat{\alpha} = E \times \widehat{\text{RSR}}(N, d, \sigma)$ and each unit-cost function is evaluated at the query N .

3.6 Validation Metrics

Model performance is reported using four metrics: coefficient of determination (R^2), root mean squared error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE). MAPE is included because runtime values span several orders of magnitude across the parameter grid.

4 Results

4.1 Real Data Properties

Table 1 summarises key statistics for the DIMACS road graphs. All eleven graphs exhibit lognormal edge-weight distributions (Figures 1 and 2). Log-scale standard deviations range from $\sigma = 0.75$ (New York) to $\sigma = 1.12$ (California), with densities $d \approx 10^{-7}$ – 10^{-5} . All graphs fall in the supercritical regime ($\bar{k} \gg 1$), making Equation (2) directly applicable.

Table 1: Summary statistics for selected DIMACS road-network graphs.

Graph	N	d	σ	$\bar{k}$
NY	264,346	1.05×10^{-5}	0.75	2.78
BAY	321,270	7.75×10^{-6}	1.07	2.49
COL	435,666	5.57×10^{-6}	1.10	2.43
FLA	1,070,376	2.37×10^{-6}	1.05	2.53
NE	1,524,453	1.68×10^{-6}	0.93	2.56
CAL	1,890,815	1.30×10^{-6}	1.12	2.46

4.2 Decrease-Key Call Number Prediction

Step 1: Phase transition. Figure 3 shows the distribution of samples across the two regimes. The subcritical regime ($\bar{k} < 1$) comprises a minority of the experimental grid; most practically relevant graphs (including all DIMACS instances) are supercritical.

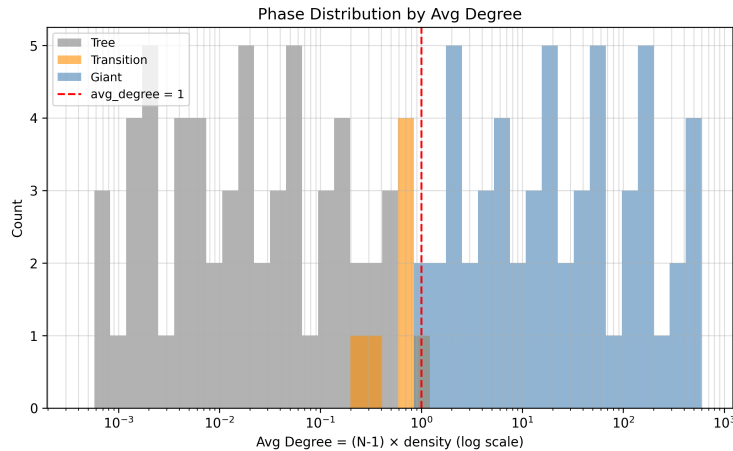


Figure 3: Distribution of (N, d, σ) samples across subcritical ($\bar{k} < 1$) and supercritical ($\bar{k} \geq 1$) regimes. The vertical dashed line marks the phase-transition threshold at $\bar{k} = 1$.

Step 2: E as a function of d and N . By construction, $E = d \cdot N(N - 1)$, yielding $R^2 = 1.000$ (Figure 4). No modelling is required for this component.

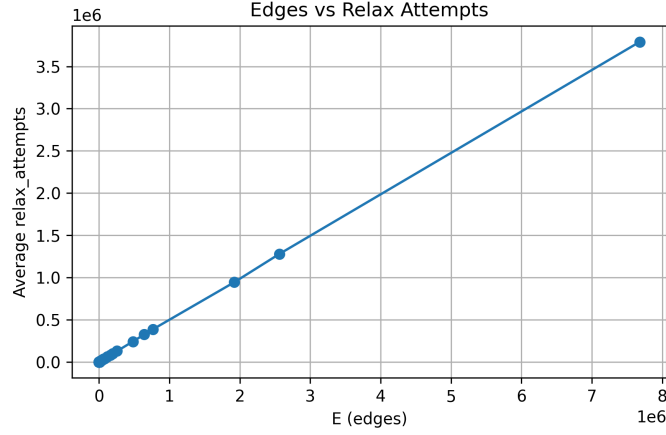


Figure 4: Total edge count E versus total relax attempts. The relationship is exact ($R^2 = 1.00$), confirming that every edge is relaxed exactly once.

Step 3: RSR model. In the supercritical regime, RSR is modelled by Equation (2). Figure 5 compares predicted and observed RSR values. The nonlinear model captures 97% of variance ($R^2 = 0.9947$), demonstrating that σ and $\bar{k}$ together explain RSR with high fidelity.

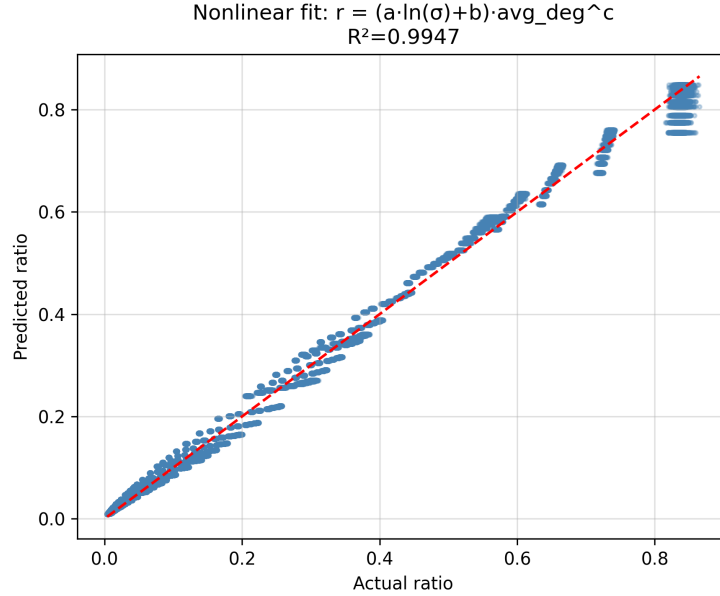


Figure 5: Predicted versus actual RSR under the nonlinear model (Eq. 2). Points cluster tightly around the diagonal ($R^2 = 0.9947$).

Step 4: Sigma effect. Figure 6 illustrates the monotone decrease of RSR with σ , holding $\bar{k}$ fixed: higher weight variance creates more opportunities for distance improvements, paradoxically resulting in fewer successful updates because early paths to most nodes are already near-optimal.

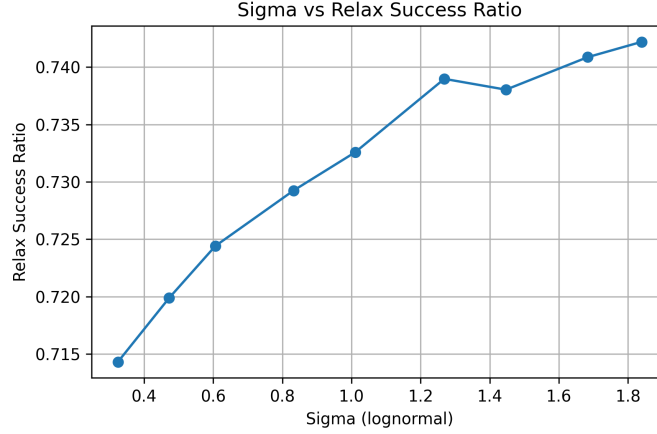


Figure 6: RSR as a function of σ (fixed $\bar{k}$). RSR decreases monotonically as σ increases, consistent with the power-law component in Equation (2).

4.3 Operation Unit Cost Analysis

Relax-attempt cost $k_3(N)$. Figure 7 shows k_3 estimates plotted against N . Despite the $O(1)$ theoretical prediction, k_3 decreases with N and levels off, well described by the exponential-decay model ($R^2 = 0.953$). This counter-intuitive finding is interpreted as a cache warm-up effect: for small N the heap fits entirely in L1/L2 cache, while for larger N cache effects approach saturation, yielding a constant marginal cost.

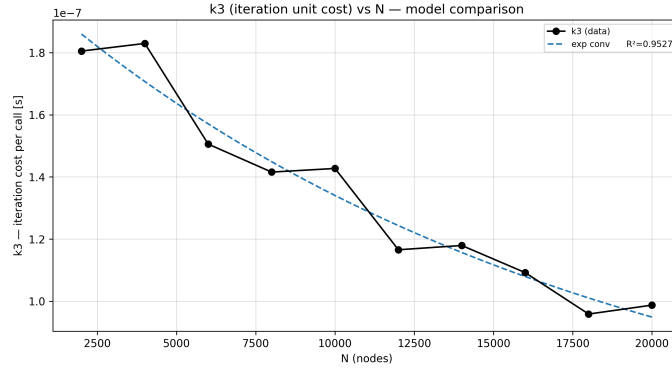


Figure 7: Per- N estimates of the relax-attempt unit cost k_3 with the fitted exponential-decay curve. The monotone decrease with N is attributed to cache effects.

Decrease-key cost $k_4(N)$. The decrease-key cost grows proportionally to $\log V$, as Figure 8 confirms ($R^2 = 0.951$). This is consistent with binary-heap theory: each decrease-key performs at most $\lceil \log_2 V \rceil$ comparisons and swaps.



Figure 8: Decrease-key unit cost coefficient k_4 as a function of N , with the fitted $a \log V + b$ curve ($R^2 = 0.951$), consistent with the $O(\log V)$ theoretical prediction.

Operation contribution. Figure 9 decomposes total runtime into three components across N . The relax-attempt term $E \cdot k_3$ contributes 55–65% of total runtime and dominates throughout. The decrease-key term $\alpha \cdot k_4 \log V$ contributes 22–44%, increasing with N as the heap height grows. The add-plus-extract term $V \cdot UC_1$ contributes only 1–15%.

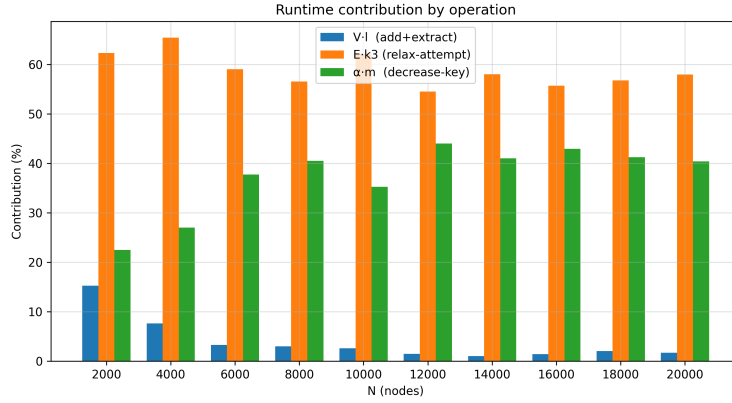


Figure 9: Fractional contribution of each operation class to total runtime as a function of N . The relax-attempt term dominates at all sizes.

4.4 Integrated Runtime Prediction

Synthetic validation. Figure 10 plots predicted against measured runtimes for all synthetic test instances. The integrated model achieves $R^2 = 0.9691$, RMSE = 57.6 ms, MAE = 35.1 ms, and MAPE = 30.9%. Predictions are reliable across most of the parameter space; the largest relative errors occur at very small N (where the intercept term is least stable) and at the phase-transition boundary.

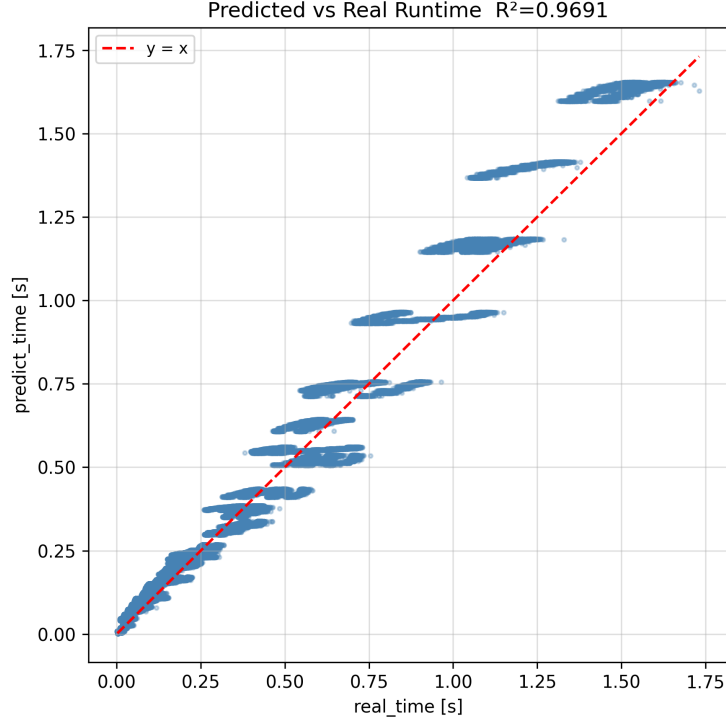


Figure 10: Predicted versus actual runtime (ms) on the synthetic test set ($R^2 = 0.9691$, $\text{RMSE} = 57.6$ ms). The dashed line is the ideal prediction ($\hat{t} = t$).

DIMACS validation. Table 2 compares predicted and measured runtimes for DIMACS graphs. Decrease-key call counts are predicted with a relative error of approximately 12% (mean across graphs), indicating that the RSR model generalises well beyond the training distribution. However, total runtime is overestimated by roughly a factor of two. Figure 11 illustrates this systematic overestimation.

Table 2: Runtime prediction on DIMACS validation graphs. Relax count relative error $\approx 12\%$; runtime overestimated by $\sim 2\times$.

Graph	N	Actual t (s)	Predicted t (s)	Runtime rel. err.	Relax rel. err.
NY	264K	1.23	2.70	1.20	0.08
BAY	321K	1.45	3.37	1.33	0.12
COL	436K	2.11	4.79	1.28	0.13
FLA	1.07M	5.88	13.67	1.32	0.11
NE	1.52M	9.24	20.36	1.20	0.11
CAL	1.89M	11.77	26.03	1.21	0.12

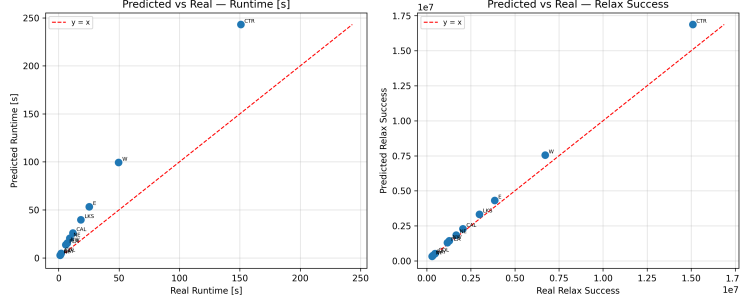


Figure 11: Predicted versus actual runtime on DIMACS validation graphs. The model systematically overestimates by $\approx 2\times$, attributed to memory-hierarchy effects not captured in training.

5 Discussion

5.1 Phase Transition as Regime Boundary

The discovery that $\bar{k} = 1$ separates two qualitatively different RSR behaviours is not accidental. Erdős and Rényi (Erdős & Rényi, 1960) proved that random graphs undergo a structural phase transition at this threshold: below $\bar{k} = 1$ the graph consists of small isolated components, while above it a giant connected component of size $\Theta(N)$ emerges. In the subcritical regime, Dijkstra visits few vertices and almost every relaxation discovers a genuinely shorter path, so $\text{RSR} \approx 1$. In the supercritical regime, the algorithm traverses a large connected component and encounters many redundant relaxation attempts, driving RSR below 1.

This structural boundary has practical implications: a separate model is necessary for the subcritical regime. Road networks universally satisfy $\bar{k} > 1$ (typical values 2–3 as shown in Table 1), so Equation (2) is directly applicable without regime detection.

5.2 Cache Effects in Unit Cost

The empirical finding that k_3 *decreases* with N contradicts the $O(1)$ asymptotic prediction. For small N , the heap is small enough to reside in CPU cache, enabling rapid access; however, the overhead of Python interpreter dispatch is proportionally larger relative to the actual heap work. As N increases, per-element memory access costs stabilise, yielding a lower effective k_3 . The exponential-decay fit captures this transition well ($R^2 = 0.953$), but the inferred curve is specific to the Python 3.12 interpreter and the 8 GB VM environment. In compiled languages or on machines with different cache hierarchies, k_3 would follow a different trajectory.

5.3 Asymptotic vs Empirical Cost

The standard complexity $O((V + E) \log V)$ implicitly assumes: (i) every relax attempt succeeds, (ii) all V decrease-keys occur, and (iii) unit costs are constant. The results challenge all three assumptions. RSR can be as low as 0.05 in dense, high-variance graphs, reducing decrease-key calls by a factor of 20 relative to the pessimistic bound. Furthermore, both k_3 and k_4 vary with N , reflecting hardware-level phenomena absent from asymptotic analysis. The decomposed model in Equation (1) captures these realities and achieves substantially better predictive accuracy than the asymptotic formula alone would permit.

5.4 Limitations

Several limitations constrain the generalisability of the results:

Language and environment. All measurements were obtained with a Python 3.12 implementation on a single Debian VM. The unit-cost coefficients k_3 and k_4 are environment-specific. The RSR model (Equation 2) is implementation-independent and should transfer across languages, but the runtime prediction requires re-fitting unit costs in any new environment.

Node-count range. Training was conducted for $N \leq 20,000$, while DIMACS graphs contain up to 23 million nodes. Extrapolation of the unit-cost curves beyond the training range introduces systematic bias, as evidenced by the $\approx 2\times$ overestimation on DIMACS.

Memory hierarchy. Training graphs fit comfortably in RAM with no paging, while large DIMACS graphs may trigger memory pressure effects. The training unit costs therefore underestimate real hardware costs at scale, resulting in overestimated predicted runtimes when the unit-cost model infers lower cost per operation than actually observed.

Graph model. Synthetic graphs follow the Erdős–Rényi random model. Real road networks possess spatial embedding, planarity, and degree-distribution properties (Singh et al., 2025) that the random model does not capture. The RSR model nonetheless generalises well to DIMACS, suggesting robustness to structural differences at this level.

6 Conclusion

6.1 Summary of Contributions

This study demonstrates that the wall-clock runtime of binary-heap Dijkstra can be predicted from three graph properties (N, d, σ) through an operation-decomposed model. Three main findings emerge:

1. **Phase transition as structural boundary.** Average degree $\bar{k} = 1$ determines the operating regime of RSR, consistent with Erdős–Rényi percolation theory. This boundary must be respected by any runtime model.
2. **Accurate RSR prediction.** The multiplicative model $\text{RSR} = (\sigma^2)^a(b \ln \bar{k} + c)$ achieves $R^2 = 0.9947$ on synthetic data and generalises to DIMACS road networks with approximately 12% relative error on decrease-key counts.
3. **Empirically calibrated unit costs.** Per-operation costs deviate systematically from theoretical $O(1)$ predictions due to cache effects, necessitating environment-specific calibration.

6.2 Practical Implications

The most practically transferable result is the RSR model. Because decrease-key call counts depend only on graph topology and weight distribution — not on implementation language or hardware — the model provides a portable tool for estimating algorithmic workload before execution. A practitioner with knowledge of N , d , and σ can compute $\hat{a}$ and assess whether Dijkstra is feasible, independent of the execution environment.

For environments where unit costs can be calibrated through short benchmark runs, the full integrated model offers runtime estimates with MAPE of approximately 31% on the synthetic domain. While this precision may be insufficient for hard real-time scheduling, it is adequate for coarse feasibility assessment and resource allocation decisions.

6.3 Future Work

Four directions merit further investigation. First, extending the training range to $N > 100,000$ would allow direct calibration at scales relevant to real road networks, potentially closing the factor-of-two gap observed on DIMACS. Second, profiling cache-miss rates alongside runtime measurements would enable explicit modelling of memory-hierarchy effects, yielding hardware-portable cost functions. Third, the subcritical regime ($\bar{k} < 1$) requires a separate model; developing a unified formulation across both regimes would broaden applicability. Fourth, extending the framework to compiled implementations (C++, Java) would test whether the RSR model transfers while confirming that only unit costs require recalibration.

References

- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269–271. <https://doi.org/10.1007/BF01386390>
- Erdős, P., & Rényi, A. (1960). On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5(1), 17–60.
- Fu, L., Sun, D., & Rilett, L. (2006). Heuristic shortest path algorithms for transportation applications: State of the art. *Computers & Operations Research*, 33(11), 3324–3343. <https://doi.org/10.1016/j.cor.2005.03.027>
- Fuhao, Z., & Jiping, L. (2009). An algorithm of shortest path based on dijkstra for huge data. *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, 4, 244–247. <https://doi.org/10.1109/FSKD.2009.848>
- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107. <https://doi.org/10.1109/TSSC.1968.300136>
- Idowu, A. I., Olabiyisi, S. O., Alo, O. O., Adeleke, I. A., Jokotoye, A. A., & Omotade, A. L. (2025). Comparative performance analysis of some priority queue variants in dijkstra’s algorithm. *International Journal of Research and Scientific Innovation*, 12(8). <https://doi.org/10.51244/IJRSI.2025.120800078>
- Singh, A., Khetarpal, R., & Rai, A. (2025). Role of spatial embedding and planarity in shaping the topology of the street networks. *Physica A: Statistical Mechanics and its Applications*, 677, 130901. <https://doi.org/10.1016/j.physa.2025.130901>
- Waga, A., Benhlilima, S., Bekri, A., Abdouni, J., & Saber, F. Z. (2025). A survey on autonomous navigation for mobile robots: From traditional techniques to deep learning and large language models. *Journal of King Saud University Computer and Information Sciences*, 37(7), 198. <https://doi.org/10.1007/s44443-025-00216-x>
- Williams, J. W. J. (1964). Algorithm 232: Heapsort. *Communications of the ACM*, 7(6), 347–348.