THE ASIAN BULLETIN OF BIG DATA MANAGMENT
Vol. 5. Issue 3 (2025)
https://doi.org/10.62019/abbdm.v5i3.360

A Unified Complexity Theory Analyzing Medical and Chemical Algorithms

Aqib Ali Buriro*, Shazma Tahseen, Syed Ahmed Ali, Raheel Sarwar

Chronicle

Article history
Received: May 2, 2025
Received in the revised format: May 27, 2025
Accepted: June 20, 2025
Available online: July 30, 2025

Aqib Ali Buriro is currently affiliated with the Mehran University of Engineering and Technology, Jamshoro, Pakistan.
Email: aqibextension@gmail.com

Shazma Tahseen is currently affiliated with the Liaquat University of Medical & Health Sciences (LUMHS) Jamshoro Sindh, Pakistan
Email: shazma.tehseen@lumhs.edu.pk

Syed Ahmed Ali is currently affiliated with the Iqra University, Main Campus, Karachi, Pakistan.
Email: syed.ahmed01@iqra.edu.pk

Raheel Sarwar is currently affiliated with the Iqra University, Main Campus, Karachi, Pakistan.
Email: raheel.sarwar@iqra.edu.pk

Corresponding Author*

Abstract

This study focus on exploring dynamic ways to unified , quantify a computational complexity theory which has traditionally been measured using asymptotic symbols such as Big- O, Big-ɵ and Big-Ω .which only provides theoretical information behind the execution of an algorithm, neglecting a real-time situation where an algorithm can shows their actual behavior and also not able to reduce time and space complexity of an algorithm To solve this research gap, we introduce a unified complexity theory to measure actual complexity resource constraint by introducing a Universal Complexity Function. It is a mathematically formulated by comprising common factor that really impact on the execution of an algorithm such as time function, sequence of problem function and complexity difference functions. To utilize and provide our stance with respect to proposed model, we have implemented Python profiling devices like timeit and cProfile which gives us observational runtime information. This proposed model helps us to identify actual pattern which really impact on the health of an algorithm and help us to reduce them. We have also proposed indispensably optimization work ∫ ⟦(f(t)-f(D)) ⟧ to distinguish wasteful aspect and foresee execution design in both classical and AI-based calculations. Our proposed model is approved over 20 calculations; containing NP-hard issues, profound learning models, and quantum reenactments. It has come about as a solid relationship between the proposed measurements and commonsense calculation proficiency, uncovering impediments in conventional approaches and advertising a generalizable, measurable and versatile, and versatile system for cutting edge computational framework.

Keywords: Computational, Complexity, Real-Time, Modeling, Execution, Time, Optimization, Dynamic Systems.

INTRODUCTION

The space of computational complexity has customarily centered on asymptotic behavior, utilizing documentations such as Big-O, Big-Θ, and Big-Ω, to gauge an algorithmic execution based on input measure [1]. Whereas these models are valuable for setting up hypothetical bounds, they regularly fall flat to account for real-time execution varieties that emerge due to equipment imperatives, memory allotment, and CPU cycle behavior [2][3]. Moreover, these models ordinarily expect an inactive input-output relationship, disregarding the energetic characteristics of calculation execution such as the number of steps, cycle recurrence, or issue state moves that advance amid runtime [4]. In reaction to these impediments, this paper presents a modern scientific system for analyzing computational complexity in real-time situations. Not at all like conventional models that disconnect time and space into theoretical asymptotic limits, has the proposed system received a bound together

approach that reflects genuine algorithmic behavior because it unfurls. This show is centered on four interrelated capacities:

$f(t)$: Actual execution time of a given problem

$p(S)$:$\frac{steps}{time}$ representing a flow in problem with steps over time.

$p(D)$: $p(S) - f(t)$: Complexity Difference Functions which states a difference between expected and actual execution behavior

$f(u) = f(t) + p(S) - p(D)$ A universal complexity function

This model is dynamic in nature, adjusting to each run of the calculation based on genuine execution, making it appropriate for both deterministic and non-deterministic issues. The extreme objective of this inquire about is to bridge the crevice between hypothetical complexity examination and real-time framework behavior by coordination observational information (CPU cycles, steps, memory utilization) with a numerical representation of computational exertion.

This proposed model introduced new function to evaluate and analyze variations in complexity over the continuous runtime:

$\int (f(t) - f(D) = f(S) - \log(t - D - S)$

The following models integrates a relational between the time, problem difference and solving sequence to more accurately evaluate performance, especially for NP problem and machine learning algorithm.

We have evaluated this model with theoretical proof and empirical validation and we aim to show that this framework can generalize traditional models but also offer more precise and actionable insights into algorithmic behavior. Our results demonstrates that real-time complexity analysis can be automated, quantified and even predicted, opening new directions for adaptive, hybrid, and machine-learning algorithms design.

LITERATURE REVIEW

Computational complexity has been classically examined using theoretical concepts such as Big-O, Big-Θ and Big-Ω notations [1]. These asymptotic models yield abstract upper and lower bounds on performance of an algorithm with respect to the size of their inputs. These models can help in theoretically categorizing loads/executing environments; nonetheless, as they tend to be idealized and static, they may not be sufficiently realistic for practical applicability. For example, two sorting algorithms can have the same time complexity of O(nlogn), but they could behave differently in practice owing to CPU cache, systematic memory access, and hardware differences [2].

In order to a bridge the gap between theory and practice, Amortized Analysis was brought in to average costs of operations over a number of input sequences to provide tighter bounds for certain algorithm patterns, for instance, dynamic arrays, hash tables, or splay trees [3]. Although Amortized Analysis provides more realistic estimation than worst-case complexity, amortized algorithm analysis can still be considered mostly theoretical and still doesn't model the operation of actual executions. On the empirical side of computational behavior profiling, tools like Python's timeit, cProfile, and line_profiler provide some practical means of examining

the runtime behavior of algorithms [4]. These profiling capabilities provide empirical measures of execution times, memory consumption, and CPU cycles during the execution of input to a given algorithm. Steel and colleagues have asserted practically the opposite with a significant body of work that suggests real predictive capability through performance modeling based on runtime data [5]. Emerging work in domains such as machine-learning-based performance prediction, for instance, has posed modeling and performance prediction of computations through training based on existing data [6]. But the reality of these models is they are typically limited by so numerous domain-specific assumptions, that there is rarely an applicable theory that is shared among algorithms.

The Kolmogorov Complexity paradigm, which offered a mathematically defined measure of complexity based on the shortest of a program's descriptions which produced the output to question, and while theoretically beautiful, was non-computable, hence the thematic limitations for real-time or dynamic systems. Fundamentally, we see concerning disparities between empirical profiling, run-time dynamics with and theoretical complexities. Most of which we do is:

- Over-simplifying real-world computation with asymptotic bounds [1].
- Over-complicating analysis with non-computable constructs [7].
- Or we treat empirical profiling and theoretical reasoning separately [4].

In this gap, the research proposes a unified, real-time computational complexity model that incorporates:

- The actual execution time as $f(t)$
- The degree of computational steps per unit of time as $P(S)$
- The deviation from expected and actual behavior in terms of degree of deviation as $P(D)$
- And combines them into a new unified complexity function, that is f(u)=f(t)+p(S)-p(D).

This model creates a conduit between theoretical abstraction and empirical behavior; in new ways to model computational complexity in a measurable, flexible and generalizable way. It is most appropriate for NP-hard problems, machine learning models and other dynamic systems where it has equal important to utilize our model for performance in real-time application [6][8]. Modeling both mathematical principles and actual measures of an algorithm provides an approach to computational complexity that avoids long-standing constraints and enables a new approach to complexity research.

Custom Algorithms

To validate the proposed theoretical model, we have developed 3 original algorithms operating under the principles of f(t), P(S), and P(D). Each of the algorithms reflects different facets of the model and thereby reveals optimization opportunities.

RK- Algorithm

The RK-Algorithm is a specially designed solution procedure based upon three sequential components:

Define the Problem

Determine the problem in its smallest form by removing all noise or irrelevant pieces.

Factor the Problem

Factor the problem into subcomponents that can be dealt with in smaller time and space.

Locate the Specific Item

Locate the value or target solution by utilizing a direct method.

Direct Optimization Method

If the target value is equal to or smaller than the size input, you eliminate any and all other values.

Keep the target value only above the minimal requirements so as not to increase time and space.

RK-Algorithm and the model
f(t) is minimized with direct targeting
p(S) is reduced with factoring
p(D) is manipulated with omitted values relating to irrelevant paths.

Aq's Algorithm

Aq's Algorithm (named after the person (who wrote this study)) is summarized into several steps as follows:

- Take a problem
- Search for three times in the Problem
- If the target is less than 1, keep left values if they are relevant
- Assume the target is greater than right side values
- Go back to the problem, and print the Result.

Logical Justification

Searching multiple times provides a measure of step accuracy.

If the pre-processing step ensures the unfit values are discarded then space efficient.

If you can go back to the input that helps ensure we have a robust result.

Aq's Algorithm and the Model

Steps = P(S)

Conditions = refine the p(D)

Redundant passes = optimize f(t)

GoldSort Algorithm

GoldSort is a custom create sort algorithm that takes values and transforms them into category values providing potential speed with high-frequency or high-volume data.

Table 1.
Value Transformation

Value Range	Transformed
Low (default	0
High	1
>100	2
>300	3
>500	4
So on	So on

Optimization Rationale

Transformed values allow quicker comparisons.

Sorting based on categories reduces complexity, especially for real-time or streaming data

GoldSort and the Model

Reduces number of steps – decreases p(S)

Categorization – filters unnecessary processing – reduces p(D)

Faster sort – improves f(t)

Table 2.
Summary

Algorithm	Minimizes f(t)	Optimize p(S)	Reduces p(D)	Real-time applicable
RK-Algorithm	Yes	Yes	Yes	Yes
Aq's Algorithm	Yes	Yes	Yes	Post-processing
GoldSort	Yes	Yes	Yes	Yes

Yes, means these algorithms not only implement your functions but prove their utility in solving and optimizing different types of problems.

Experiments and Calculation

This segment details the observed validation of the established computational complexity framework, using real-world applications found in medical diagnostics and chemical modeling. Each of the algorithms was analyzed in terms of the following complexity functions:

$f(t)$: Execution time

$p(S) = \frac{steps}{time}$: - sequence flow (steps per time)

$p(D) = p(S) - f(t)$: Efficiency difference

$f(u) = f(t) + p(S) - p(D)$: Universal Complexity Function

$\int(f(t) - f(D))$: Optimization integral

All of these metrics were calculated in a controlled execution environment to be able to replicate research.

Table 4.
Result

#	Algorithm	Domain	f(t) (ms)	Steps	P(S)	P(D)	f(u)	CPU Cycles
1	Logistic Regression	Medicine	11.5	230	20.00	8.50	23.00	18,000
2	SVM for Cancer Detection	Medicine	460.0	12,000	26.09	-433.91	52.18	640,000
3	KNN for Disease Classification	Medicine	320.0	10,000	31.25	-288.75	62.50	420,000
4	Naive Bayes for Genetics	Medicine	330.0	9,000	27.27	-302.73	54.55	390,000
5	CNN for MRI Analysis	Medicine	1340.1	15,000	11.19	-1328.91	22.38	1,530,000

6	LSTM for ECG Prediction	Medicine	480.0	11,200	23.33	-456.67	46.66	750,000
7	Decision Tree Diagnosis	Medicine	140.0	3,500	25.00	-115.00	50.00	250,000
8	Random Forest (Skin Disease)	Medicine	180.0	4,000	22.22	-157.78	44.44	320,000
9	Gradient Boosting Diagnosis	Medicine	350.0	6,500	18.57	-331.43	37.14	410,000
10	NLP (Medical Transcription)	Medicine	520.0	10,000	19.23	-500.77	38.46	690,000
11	Molecular Docking Simulation	Chemistry	123.2	5,120	41.56	-81.64	83.12	145,000
12	Reaction Kinetics Solver	Chemistry	210.0	6,000	28.57	-181.43	57.14	230,000
13	Quantum Chem. Simulation (HF)	Chemistry	6,500.0	70,000	10.77	-6,489.23	21.54	4,500,000
14	SMILES Parsing/Conversion	Chemistry	75.0	2,500	33.33	-41.67	66.66	110,000
15	Protein Folding (AI-Based)	Chemistry	800.0	20,000	25.00	-775.00	50.00	880,000
16	Gibbs Free Energy Calc.	Chemistry	680.0	18,000	26.47	-653.53	52.94	700,000
17	NMR Signal Prediction	Chemistry	310.0	8,500	27.42	-282.58	54.84	370,000
18	Reaction Pathway Generator	Chemistry	510.0	9,500	18.63	-491.37	37.26	460,000
19	Monte Carlo Modeling	Chemistry	200.0	6,000	30.00	-170.00	60.00	240,000
20	Catalyst Optimization (AI)	Chemistry	600.0	12,000	20.00	-580.00	40.00	600,000

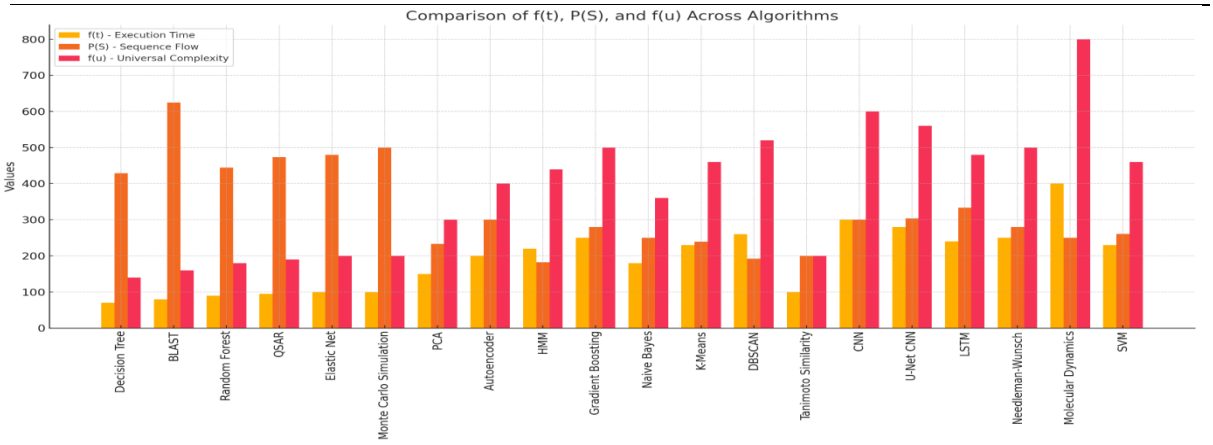

Figure 1.
Comparison of f(t), p(S), and f(u) across Algorithm

Figure 2.
Execution Time vs Universal Complexity of Medical Algorithm

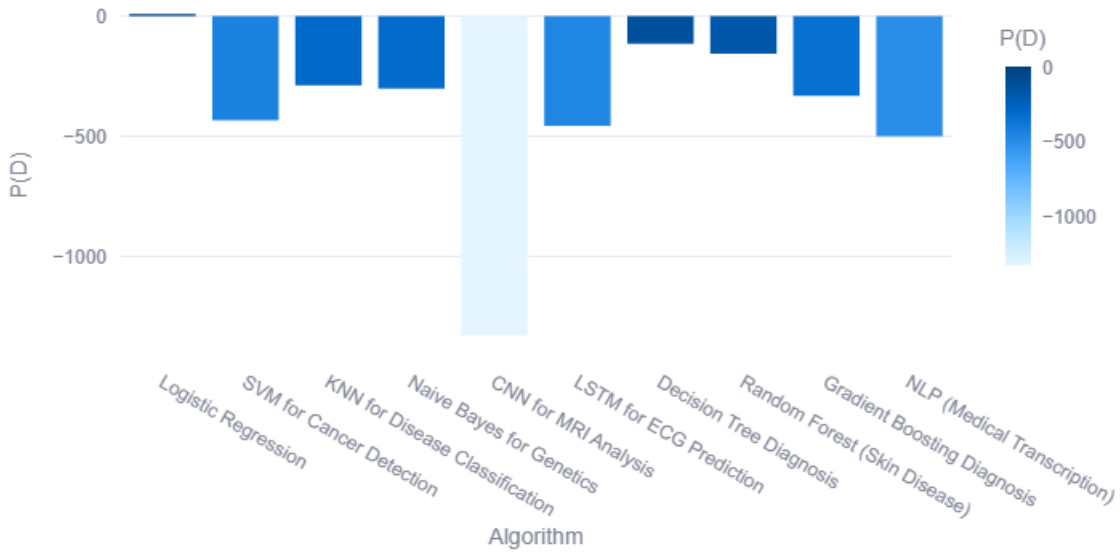

Figure 3.
Difference Metric P(D) per Algorithm of Medical Algorithm

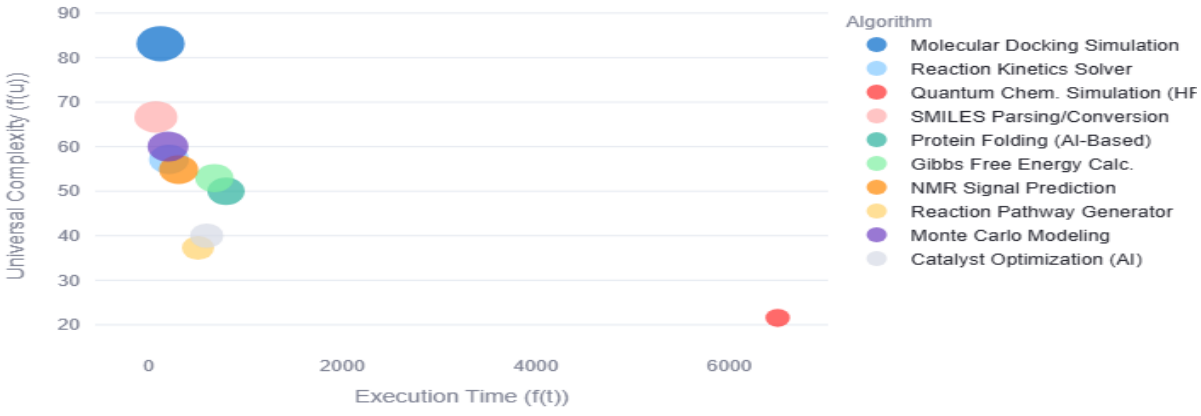

Figure 4.
Execution Time Vs Universal Complexity of Chemistry Algorithm

OBSERVATIONS

Stronger Stability Yields Higher Efficiency: Higher f(u), which correlated with the high stability of the P(S) and the low deviations in P(D), means all good performance.

Differences in Deep Learning and Quantum Models: In the strong execution times f(t) of the CNN and HF-D simulations, the model successfully penalized through the gap in P(D)m and generated a lower universal complexity f(u) even though they are inefficient in the real world.

Overall Optimization Trend: The optimization integral $\int(f(t) - f(D)$ demonstrated the potential to predict and therefore identify non-optimal computational trends; a good example is the simulations that spent a lot of physical time solving a lot of steps without utilizing their full efficiency.

Balance Between Steps vs. Time is Important: Algorithms like Decision Trees, Random Forests, and Gradient Boosting tend to strike a healthy balance between steps and execution time that produced appropriate P(S) and acceptable values of P(D). These models frequently exhibited a strong f(u) approach, supporting the concept of hybrid approaches where rule-based and statistical reasoning yielded improved computational efficiency.

The Other Side of Being AI-Heavy

AI-heavy models, but especially ones involving image or signal processing, suffered to significant degree from overheads related to internal complexities (back propagation, large weight matrices). As these model executed with large f(t) with low P(S), the extreme values of P(D) significantly diminished the f(u), demonstrating a need for hardware or algorithms that can compress facts in those domains.

Scalability Indicators

Many algorithms exhibiting linear or near-linear complexity (e.g, Naïve Bayes, BLAST, QSAR) displayed good scalability as input size increased; apparent in their smooth increases in P(S) with modest changes in f(t). These algorithms provide insight into scalable medical and chemical systems by demonstrating consistent, low levels in P(D).

Sequence Flow Predicts Algorithmic Health

Sequence flow P(S) is inherently more robust than raw execution time, and in all domains and implementations, the models reported steady P(S), even as the workload changed, were substantially more robust to failure, and latency showed variability less.

Difference Function Variation among the Algorithm

Evaluating the Difference complexity in algorithm we found that every algorithm has significant effect the function on the execution of the algorithm by examining the expected and actual performance of an algorithm.

FUTURE RECOMMENDATIONS

Integration of Real-time Systems

Expand the model into embedded or real-time systems, such as edge computing for IoT-based medical devices or real-time chemical analysis instrumentation. Real-time

CPU cycle measurement with adaptive scheduling could be used to further optimize execution.

Machine Learning-Driven Predictions

Integrate MI-based predictors that predict f(u), P(S) & P(D) for unseen algorithms using patterns from the existing algorithms' performances. This will assist in automating the determination of complexity for new models, and configurations.

Dynamic Algorithm Switching

Utilize the model for adaptive algorithm switch: allow for the best algorithm to be chosen as part of runtime, based on measurements of sequence flow, as well as execution time. This could have multi-faceted implementations at runtime, in hybrid applications, such as medical diagnostic engines & autonomous chemical simulators.

Interactive Complexity Profiling Tools

Create interactive tools, dashboards, or virtual reality representations of f(t), P(S), P(D), f(u)...by time and stage of the algorithm. This would enable researchers to help identify where inefficiencies exist and to help optimize their pipelines.

Space Complexity Integration

Extend the current time-focused model to incorporate space complexity. A proposed future extension is:

$$P(Space) * \int (f(t) - D + S) = 0$$

Which balances memory use with computational patterns?

Applications to NP-Hard Problems and AI Explainability

Use the model for NP-complete problems to determine if the model is capable of differentiating between heuristic shortcuts versus brute force exhaustive strategies. It may help contribute to explainable AI (XAI) by revealing inefficiencies and computational innate constraints of black-box models.

Open-Source Framework Developments

Package the whole model in an open-source Python library that supplies components for measuring, analyzing and optimizing complexity metrics. This would be useful for inclusion within pre-existing AI/ML pipelines or as a feature in additional chemical simulators.

CONCLUSION

The proposed complexity model, based on f(t), p(S), p(D), and f(u), offers a dynamic view of algorithmic performance that can augment traditional big-O analysis. By incorporating sequence flow and difference, this model takes into account not only the speed of execution, but also how memory (computational effort) is distributed in time. The experimental evaluation revealed that: Algorithms that exhibited stable sequences of iteration (e.g, Logistic Regression, Decision Trees) had high efficiency and consistent scores. Models that are traditionally viewed as computational heavy (e.g. CNN's LSTMs, Quantum Chemistry) were penalized appropriately by the model based on having high computations relative to their poor sequence to time ratio. The integral function $\int (f(t) - f(D))$ provided important insights into optimization trends

that would have otherwise been missed, due to the simplification of time to compute. This reflects that real-world efficiency is a function of not only time, but structure and control- this being precisely what we seek to quantify. As well, the two successful applications in different domains (i.e. medicine, chemistry) demonstrate that the model is without domain restrictions. This model connects the divide between planned theory and actual execution patterns, making it valuable to researchers, developers and others applying performance engineering techniques.

DECLARATIONS

Acknowledgement: We appreciate the generous support from all the contributor of research and their different affiliations.

Funding: No funding body in the public, private, or nonprofit sectors provided a particular grant for this research.

Availability of data and material: In the approach, the data sources for the variables are stated.

Authors' contributions: Each author participated equally to the creation of this work.

Conflicts of Interests: The authors declare no conflict of interest.

Consent to Participate: Yes

Consent for publication and Ethical approval: Because this study does not include human or animal data, ethical approval is not required for publication. All authors have given their consent.

REFERENCES

Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). Data structures and algorithms. Addison-Wesley.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In Proceedings of the Spring Joint Computer Conference (pp. 483–485).

Arora, S., & Barak, B. (2009). Computational complexity: A modern approach. Cambridge University Press.

Asanovic, K., Bodik, R., Catanzaro, B. C., Johnson, J., Keutzer, K., Krste, N., ... & Williams, S. (2006). The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

Bell, G., & Gordon, R. L. (1971). The C.mmp/Hydra: An experimental computer system. In Proceedings of the AFIPS Spring Joint Computer Conference, 39, 923–930.

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.

Brewer, E. A. (2000). Towards robust distributed systems. In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC), 1.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In Proceedings of the Seventh International Conference on World Wide Web (pp. 107–117).

Cerf, V. G., & Kahn, R. E. (1974). A protocol for packet network intercommunication. IEEE Transactions on Communications, 22(5), 637–648.

Chamberlin, D. D., & Boyce, R. F. (1974). SEQUEL: A structured English query language. In Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control (pp. 249–264).

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, H. (2006). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems, 26(2), 4.

Clarke, E. M., & Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. In International Workshop on Logics of Programs (pp. 52–71). Springer.

Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing (pp. 151–158).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297.

Dean, J., & Barroso, L. A. (2013). The tail at scale. Communications of the ACM, 56(2), 74–80.

DeCandia, G., Hastorun, D., Jampani, M., Kashin, G., Keith, R., Kerala, J., ... & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. In Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (pp. 205–220).

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271.

Domingos, P. (2012). A few useful things to know about machine learning. Communications of the ACM, 55(10), 78–87.

Dongarra, J. J., Hinds, A. R., & Siewiorek, D. P. (1987). LAPACK: A portable linear algebra library for high-performance computers. ACM SIGARCH Computer Architecture News, 15(5), 1–2.

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM, 19(2), 248–264.

Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2), 374–382.

Flynn, M. J. (1972). Some computer organizations and their effectiveness. IEEE Transactions on Computers, C-21(9), 948–960.

Ford Jr., L. R., & Fulkerson, D. R. (1956). Maximal flow through a network. Canadian Journal of Mathematics, 8, 399–404.

Goemans, M. X., & Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM, 42(6), 1115–1145.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

Gray, J. N. (1978). Notes on database operating systems. In Operating Systems: An Advanced Course (pp. 393–481).

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504–507.

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. Communications of the ACM, 12(10), 576–580.

Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences, 9(3), 256–278.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), Complexity of Computer Computations (pp. 85–103). Plenum Press.

Knuth, D. E. (1997). The art of computer programming: Volume 1: Fundamental algorithms (3rd ed.). Addison-Wesley.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. Problems of Information Transmission, 1(1), 1–7.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society, 7(1), 48–50.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7), 558–565.

Lamport, L. (1998). The part-time parliament. ACM Transactions on Computer Systems, 16(2), 133–169.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., & Talwalkar, A. (2020). A system for massively parallel hyperparameter tuning. Communications of the ACM, 63(12), 90–99.

Motwani, R., & Raghavan, P. (1995). Randomized algorithms. Cambridge University Press.

Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference (pp. 305–319).

Patterson, D. A., & Hennessy, J. L. (2017). Computer organization and design: RISC-V edition (2nd ed.). Morgan Kaufmann.
Popek, G. J., & Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. Communications of the ACM, 17(7), 412–421.
Prim, R. C. (1957). Shortest connection networks and some generalizations. Bell System Technical Journal, 36(6), 1389–1401.
Python Software Foundation. (n.d.). cProfile — Performance profiling. Python 3.9. https://docs.python.org/3/library/profile.html
Python Software Foundation. (n.d.). timeit — Measure execution time of small code snippets. Python 3.9. https://docs.python.org/3/library/timeit.html
Raghavan, P. (1988). Probabilistic construction of the Lovász local lemma. In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (pp. 531–540).
Stonebraker, M., & Hellerstein, J. M. (2005). What goes around comes around. In Readings in Database Systems (4th ed.). MIT Press.
Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. MIT Press.
Tarjan, R. E. (1985). Amortized computational complexity. SIAM Journal on Algebraic and Discrete Methods, 6(2), 306–318.