On Upward Book Embeddability of DAGs

Rustem Kakimov* Xing Tan*†

Abstract

The k-page upward book embedding problem ($kUBE$) for directed acyclic graphs (DAGs) is of significant relevance to network visualization and circuit design. It is known to be NP-complete for $k \geq 2$. This study investigates $kUBE$ embeddability and its relationship to vertex count, edge density, and page count. We begin by exhaustively enumerating all small DAGs to identify embeddability patterns, and then extend the analysis to larger instances via representative sampling. Our results confirm that embeddability decreases with increasing graph density and improves with the number of pages. By encoding $kUBE$ as a Boolean satisfiability problem, we demonstrate the effectiveness of SAT-based methods in addressing this computationally challenging class of problems.

1 Introduction

The book embedding problem (BE) involves arranging a graph's vertices along a spine and distributing edges across k pages without crossings on the same page [3]. The k-page upward book embedding problem ($kUBE$) for directed acyclic graphs (DAGs) requires edges to be oriented upward, relevant for applications like network visualization and circuit routing [7, 8]. While $1UBE$ is solvable in linear time [13], $kUBE$ is NP-complete for $k \geq 2$ [1, 5].

We investigate $kUBE$ embeddability of DAGs, focusing on the effects of vertex count (n), edge count (m), and page count (k). Our research addresses:

- **Impact of Graph Density** As edge density (m/n) increases for fixed n, does the percentage of k-page embeddable DAGs decrease, and can we observe a phase transition?
- **Effect of Pages** How does embeddability change as k increases beyond 2?

We start with small DAGs (n small), enumerating all DAGs on n vertices, to identify any embeddability patterns. We then consider larger DAGs (obtained via sampling). Our analysis reveals density-driven phase transitions and page-dependent patterns. Using a SAT-based approach with the SAT-1 encoding [15] and the SAT solver Kissat [4], we efficiently verify embeddability, providing insights into the complexity of $kUBE$.

*Department of Computer Science, Lakehead University, Canada. {rkakimov, xing.tan}@lakeheadu.ca

†Corresponding author.

2 Preliminaries

This section outlines the core concepts and techniques used in our study. We introduce book embedding and its upward variant for DAGs, describe methods for enumerating small DAGs and sampling random DAGs, and explain the SAT-based approach for verifying embeddability, which underpins our experimental investigations.

2.1 Book Embedding

Definition 1 ***k-Page Book Embedding for Undirected Graphs*** *A book embedding of a graph $G = (V, E)$ consists of*

1. *a linear ordering $\pi : V \to \{1, 2, \ldots, |V|\}$ of vertices along the line called the spine of a book, and*
2. *an assignment $\sigma : E \to \{1, \ldots, k\}$ of individual edges to one of the k pages such that no edges assigned to the same page geometrically cross. More precisely, for any two edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$ on the same page, such that $\pi(u_1) < \pi(v_1)$ and $\pi(u_2) < \pi(v_2)$, the following two conditions are not allowed: $\pi(u_1) < \pi(u_2) < \pi(v_1) < \pi(v_2)$ and $\pi(u_2) < \pi(u_1) < \pi(v_2) < \pi(v_1)$.*

Definition 2 ***k-Page Upward Book Embedding for Directed Graph*** *An upward book embedding of a directed graph $G = (V, E)$ also consists of a linear ordering $\pi : V \to \{1, 2, \ldots, |V|\}$ of vertices along the spine of the book, and an assignment $\sigma : E \to \{1, \ldots, k\}$ of edges to k pages such that no two edges geometrically cross. Additionally, all the edges should be oriented in the same direction. That is, for any edge (u, v), $\pi(u) < \pi(v)$ must hold.*

Definition 3 ***kBE Problem*** *A k-Page Book Embedding Problem (kBE) is defined as follows: Given an undirected graph G, does there exist a k-page book embedding for G?*

Definition 4 ***kUBE Problem*** *A k-Page Upward Book Embedding Problem ($kUBE$) is defined as follows: Given a directed graph G, does there exist a k-page upward book*

embedding for G? Note that, an upward book embedding can possibly exist in a directed graph G only if the graph is acyclic. In other words, the graph G must be a DAG.

Figure 1 presents an example of a *kUBE* problem instance with three pages (i.e., $k = 3$). The input is a DAG with 6 nodes and 15 edges, shown in Figure 1a. The goal is to embed it into three pages. The resulting solution is shown in Figure 1b, where the nodes are placed in a topological order along a horizontal spine, and edges—colored to indicate page assignment—are drawn without crossings within the same page.

Both the kBE and kUBE problems are computationally challenging, as indicated in the following theorems.

Theorem 1 *For any $k \geq 2$, kBE Problem is NP-complete. Meanwhile, 1BE can be solved in $O(n)$ time. This result is due to Wigderson [25].*

Theorem 2 *For any $k \geq 2$, kUBE is NP-complete. Meanwhile, 1UBE can be solved in $O(n)$ time. These results are due to Bekos et al. [1] and Heath and Pemmaraju [13], respectively.*

2.2 Enumerating All Small DAGs

We use an algorithm that generates all DAGs for n nodes, see Algorithm 1. It begins by initializing the node set $V = \{0, 1, \ldots, n-1\}$ and the set E_{all} of all possible directed edges between distinct nodes. The algorithm then iterates over all subsets of E_{all}, constructs the corresponding graph, and checks for acyclicity via topological sort. Subsets that form valid DAGs are collected.

Given the exponential number of edge subsets—reflecting the inherent complexity of enumerating all DAGs, the algorithm adopts a brute-force approach. Each DAG verification requires $O(n^2)$ time, resulting in exponential time complexity overall. Additionally, storing all DAGs demands exponential space, limiting practical use to $n \leq 10$. Nonetheless, our implementation emphasizes simplicity and completeness. By exhaustively exploring edge combinations, it ensures every DAG is generated, making it well-suited for small n or theoretical analysis where clarity and correctness outweigh efficiency considerations.

Algorithm 1 Generate all DAGs with n nodes

Initialization : dags $\leftarrow$ [ ]; $V \leftarrow \{0, 1, \ldots, n-1\}$; $E_{\text{all}} \leftarrow \{(u, v) \mid u \neq v, u, v \in V\}$
for all edge subsets E' of E_{all} **do**
    $G \leftarrow \text{CreateGraph}(V, E')$
    If (IsDAG(G)) dags $\leftarrow$ dags $\cup \{E'\}$
end for
return dags

2.3 Generating Random DAGs via Sampling

When n is large, generating and analyzing all possible DAGs becomes computationally infeasible due to the exponential growth of the DAG instance space. Instead, we sample a subset of DAGs to study their properties, making the approach computationally manageable. It is crucial, however, to ensure that the sampled subset is representative, as an inappropriate sampling method can skew results and misrepresent the underlying properties of the DAG population. For example, a biased sampling method might over-represent certain structures—such as those with more valid topological orderings—leading to inaccurate conclusions about phenomena like satisfiability transitions.

To balance computational efficiency and statistical precision, we adopt a two-tiered strategy that integrates existing methods with our own adaptations into a cohesive framework. First, we propose Algorithm 2, an efficient topological-order-based heuristic algorithm that generates a broad set of DAGs across all edge counts m (ranging from 0 to $n(n-1)/2$) for each n. We then complement this with targeted uniform sampling in the critical phase-transition region using the Kuipers–Moffa method [19]. The combination of random sampling for broad coverage and uniform sampling for key areas ensures both efficiency and accuracy in our experimental analysis.

Algorithm 2 Generate a random DAG of n nodes and m edges

Require: n, m
Ensure: A list of m edges forming a random DAG
order $\leftarrow$ Shuffle($[0, 1, \ldots, n-1]$) ▷ Random permutation of nodes
all_edges $\leftarrow \{(order[i], order[j]) \mid 0 \leq i < j < n\}$ ▷ All forward edges
chosen_edges $\leftarrow$ UniformSampling(*all_edges*, m) ▷ Select m edges uniformly
return Sort(*chosen_edges*) ▷ Return sorted edge list

The topological-order method, detailed in Algorithm 2, generates DAGs by shuffling node orders and uniformly selecting m edges from all possible forward edges, producing 100 DAGs per m (or 30 for high n and k). Meanwhile, the Kuipers–Moffa method, implemented via the `unifDAG` R package [20], ensures uniform sampling by recursively constructing DAGs through outpoint removal and reverse connection sampling. We specifically use the approximate method for broad exploration across all m, reducing to 30 samples for computationally intensive cases ($n = 20, k \geq 6$), and validate its empirical adequacy against uniform sampling for selected n. We switch to Kuipers–Moffa sampling in the phase-transition region, where satisfiability is about

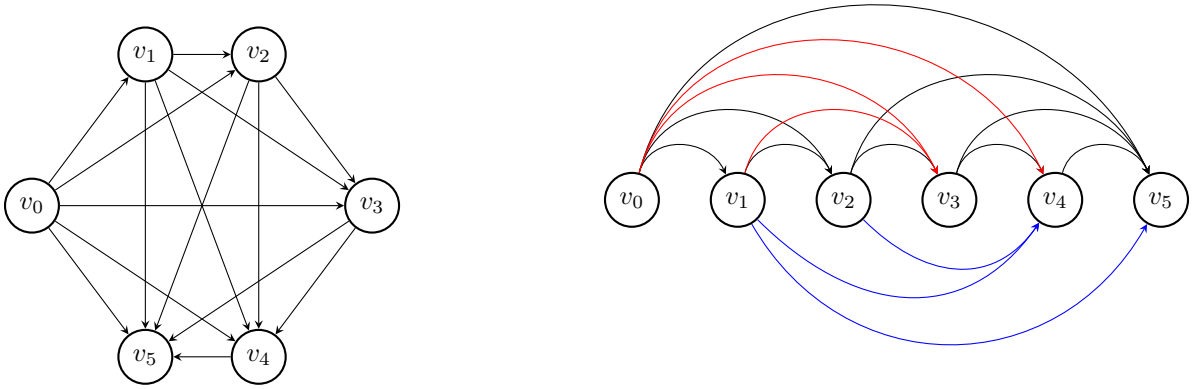


Figure 1: (a) A DAG with 6 nodes and 15 edges (maximal). (b) Its 3-page Upward Book Embedding (*3UBE*), where edges are partitioned into three pages highlighted in black, blue, and red.

50%. In this study, we used this sampling approach to generate DAGs for $n \in \{7, 8, \ldots, 20\}$, storing them as edge lists for analysis.

2.4 SAT for Embeddability Verification

To investigate the embeddability of DAGs in the *kUBE* problem, we utilize SAT-1, a specific SAT encoding for *kUBE* [15]. SAT-1 has its roots in the SAT encoding originally proposed by Bekos et al. [1, 2] for undirected book embedding problems. Several revisions have been made to address the unique requirements of linear ordering of vertices and directed edges in kUBE problems. That is, it requires a linear ordering $\pi : V \to \{1, \ldots, |V|\}$ of vertices along a spine and an assignment $\sigma : E \to \{1, \ldots, k\}$ of edges to k pages, such that edges are upward (i.e., $\pi(u) < \pi(v)$ for all $(u, v) \in E$) and no same-page edges cross geometrically.

SAT-1 transforms these constraints into a Boolean satisfiability formula called $\mathcal{F}_1(G, k)$ in conjunctive normal form (CNF), enabling efficient verification with modern SAT solvers. For a DAG $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, SAT-1 defines three key variable sets:

1. $L(v_i, v_j)$ for vertex ordering, which is true if $\pi(v_i) < \pi(v_j)$;
2. $EP(e_i, p)$ for edge-to-page assignments, which is true if edge e_i is on page $p \in \{0, \ldots, k-1\}$; and
3. $X(e_i, e_j)$ for same-page edge pairs, which is true if e_i and e_j share a page.

The formula is satisfiable if and only if G admits a valid k-page upward embedding, with a polynomial size of $O(n^2 + m^2 + mk)$ variables and $O(n^3 + m^2)$ clauses.

SAT-1's strength lies in its completeness and tailored adaptation to *kUBE*'s directed nature. The encoding for *kUBE* enforces topological ordering via clauses $L(u, v)$ for each $(u, v) \in E$, ensuring upward directionality. Compared with the original encodings [1, 2], edge-crossing constraints are simplified by considering only direction-consistent permutations (e.g., (a, c, b, d) or (c, a, d, b) for edges (a, b) and (c, d)), reducing clause overhead while maintaining correctness [1]. Transitivity of the vertex order is enforced through CNF clauses like $[\neg L(v_i, v_j), \neg L(v_j, v_k), L(v_i, v_k)]$, ensuring a consistent linear arrangement. This adaptability, combined with the efficiency of solvers, makes SAT-1 a practical tool for embeddability testing.

We have conducted empirical evaluations of SAT-1, providing strong evidence of its efficiency based on benchmarks such as the North dataset (1277 DAGs, [12]) and large grid graphs. In our current research, we apply SAT-1 to systematically assess *kUBE* embeddability across DAGs of varying sizes, densities, and structural properties. This enables us to probe phenomena such as phase transitions, the effect of increasing k and potentially other structural factors influencing *kUBE* embeddability.

2.5 Kissat: An Efficient SAT Solver

In this study, we use Kissat [4, 10, 11], a modern SAT solver, to handle SAT-1. Kissat follows the conflict-driven clause learning (CDCL) framework and incorporates optimizations such as inprocessing and phase saving to improve efficiency. Its ability to handle large constraint systems with structured variable dependencies makes it particularly suitable for satisfiability problems arising in graph theory, where constraints often reflect structural properties of graphs. In our case, Kissat

is used to determine the embeddability of DAGs under *kUBE* constraints by efficiently exploring large search spaces and resolving complex logical dependencies.

3 Upward Book Embeddability of DAGs

This section examines the upward book embeddability of DAGs, analyzing small enumerable instances, theoretical page bounds, larger random samples, and phase transitions, to characterize how graph size, edge density, and page numbers influence embeddability.

3.1 Embeddability of Small DAGs

We begin by investigating the upward book embeddability of DAGs with at most $n = 6$ vertices. In these small cases, all possible DAGs can be exhaustively enumerated. This examination provides significant insight and valuable perspectives on the broader context of larger problems.

We assess the embeddability of all enumerated DAGs for $n = 4, 5, 6$ using the SAT approach described in Section 2.4. For each n, we generate all possible DAGs with Algorithm 1, categorize them by edge count m (as shown in Tables 2, 3, and 4), and test each instance for k-page upward book embeddability with $k = 1$ and $k = 2$. The SAT-1 encoding translates a *kUBE* problem into a Boolean satisfiability problem, which is then solved using a SAT solver. For each combination of n, m, and k, we compute the percentage of satisfiable DAGs by dividing the number of SAT outcomes (indicating embeddability) by the total number of DAGs at that m, providing the data points to plot the curves in Figure 2.

Figure 2 illustrates the percentage of DAGs that can be embedded using k pages as a function of the total number of edges m, for varying numbers of nodes n. The x-axis represents m, ranging from 0 to 15, while the y-axis shows the percentage of satisfiable DAGs, from 0% to 100%. Note that when $n = 6$, the maximal possible m value is 15; when $n = 5$, the maximal m value is 10; and when $n = 4$, the maximal m is 6. The data is categorized by n and k: filled circles denote $n = 6$, crosses denote $n = 5$, and triangles denote $n = 4$, with blue representing $k = 1$ and orange representing $k = 2$.

For all configurations, we have the following observations: 1) The percentage starts at 100% when $m = 0$, as a DAG with no edges is trivially embeddable. As m increases, the percentage decreases steadily and smoothly, reflecting the growing complexity of the DAG. For any specific curve with a fixed n, this indicates that embeddability decreases as the graph density (defined as m/n) increases. 2) For a fixed n, the percentage of satisfiable DAGs is consistently higher for $k = 2$ (orange) than for $k = 1$ (blue). For instance, the solid orange line ($n = 6, k = 2$) remains above the solid blue line ($n = 6, k = 1$) across all m, indicating that using two pages provides greater flexibility to satisfy the DAG, and 3) When n is fixed and k is repeatedly increased (i.e., the total number of pages allowed for the embedding), there exists a threshold k value at which all DAGs of size n are embeddable. For small DAGs (where $n = 4, 5, 6$), these threshold k values are obtained using the SAT-1 encoding and a SAT solver, and are presented in Table 1.

Table 1: Minimal number of pages k required for all size-n DAGs to be embeddable ($n = 4, 5, 6$).

DAG size (n)	4	5	6
Minimal # of pages required (k)	2	3	3

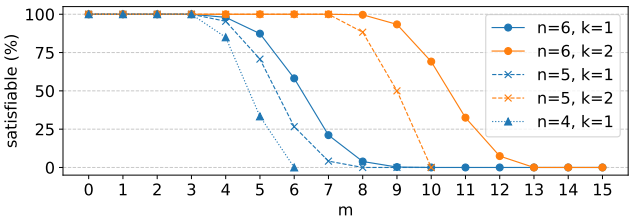


Figure 2: Percentage of satisfiable DAGs as a function of the number of edges m, for different numbers of nodes n and pages k. Filled circles represent $n = 6$, crosses represent $n = 5$, and triangles represent $n = 4$. Blue denotes $k = 1$, and orange denotes $k = 2$.

The table explains why there are only five curves in Figure 2. In addition, we also manually proved that when $k \geq 3$, all $n = 6$ DAGs are embeddable in the following theorem.

Proposition 3 *All $n = 6$ DAGs can be upward book-embedded using 3 pages.*

Proof. We know that for complete cases (where each node has an edge to every node to its right), we are able to find a 3-upward book embedding, as shown in the right subfigure of Figure 1. This implies that any $n = 6$ DAG contains fewer edges than the "maximal DAG", can be embedded using 3 pages. We conclude that all $n = 6$ DAGs can be upward book-embedded using three pages. □

3.2 Embeddability of Random DAGs

When n increases, enumerating all DAGs quickly becomes impractical. For instance, when $n = 10$, the total number of DAGs is approximately 4.18×10^{18} ([24, A003024]). To explore the embeddability properties of larger DAGs, we rely on sampling techniques, which are

introduced in Section 2.3, to generate random DAG instances. We used this method to obtain 4,600 DAGs with $n = 10$, 10,600 DAGs with $n = 15$, and 19,100 DAGs with $n = 20$, the latter of which was downsampled to 5,730 instances for $k \geq 6$.

We then benchmarked SAT solver runtimes across increasing values of k. The experimental setup covered the following configurations, with the number of edges m ranging from 0 to its maximum in all cases: (a) $n = 10$, with $k = 2, 3, 4$; (b) $n = 15$, with $k = 2$ to 7; and (c) $n = 20$, with $k = 2$ to 7.

Due to the prohibitively high computational cost at these values, results for $k = 8$ and up are not included in the current paper. For each value of n, we analyzed how increasing the edge count m, and thus the graph density m/n, impacts both the embeddability of DAGs in *kUBE* and the runtime of the SAT solver. Results are presented in Figure 3, where each subfigure corresponds to a fixed n and consists of two aligned plots: the top shows the satisfiability rate as a function of edge density m/n, and the bottom shows the mean run time of the SAT solver. In both plots, the x-axis is divided into bins of width 0.2, and the mean is calculated per bin.

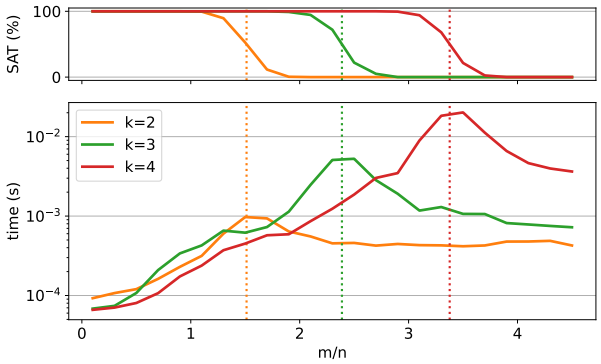
Recall that (Section 3.1) when DAGs are small and full enumeration was feasible, we identified two key patterns: 1) embeddability starts at 100% with $m = 0$, dropping smoothly as m increases due to the increasing density of the graph (m/n), and 2) higher k values, such as $k = 2$ versus $k = 1$, consistently improve embeddability by offering greater flexibility. Interestingly, these patterns hold in our experiments with sampled larger DAGs at $n = 10$, 15, and 20, where we observe a comparable gradual decline in embeddability as edge density grows, alongside a significant improvement in embedding success with larger k values, closely mirroring the behavior of their smaller, fully enumerated counterparts.

The bottom plots of the three subfigures, which depict mean computation time, exhibit a consistent pattern across all configurations of n and k:

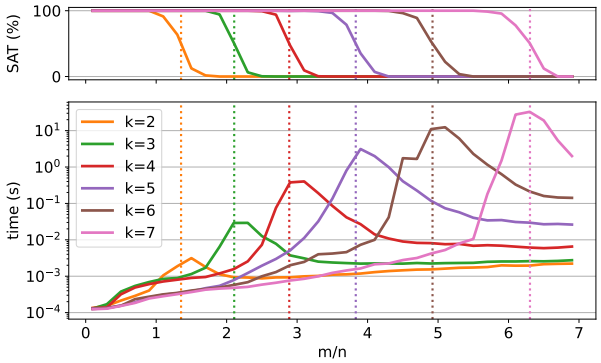
1. For each (n, k) combination, the peak runtime is aligned with the 50% embeddability threshold (indicated by the vertical dotted line). This suggests that computational complexity is maximized when the proportion of SAT to UNSAT instances is approximately balanced.

2. For larger values of k, the peak runtime increases substantially compared to smaller k. As the y-axis uses a logarithmic scale, we can observe that the peak runtime for k is typically one order of magnitude higher than that for $k - 1$.

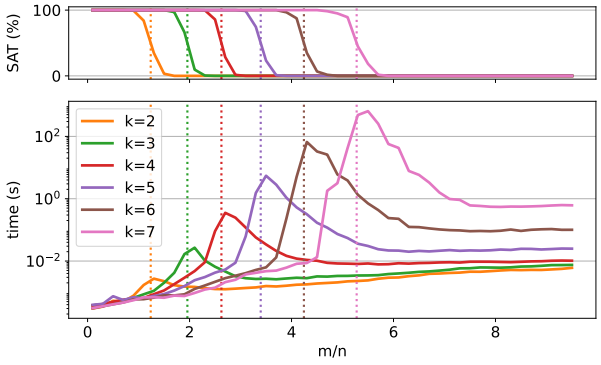
3. Prior to the peak, the runtime differences across k values are relatively small. Notably, there is consistently a range of m/n values where higher k in-



(a) $n = 10$



(b) $n = 15$



(c) $n = 20$

Figure 3: Embeddability and computational cost for sampled DAGs with $n = 10$, 15, and 20. Each subfigure consists of: (Top) Fraction of embeddable DAGs versus edge count m, plotted for varying k values, showing a consistent decline with increasing m. (Bottom) Mean computation time aligned with m, displaying a peak near the 50% embeddability threshold (vertical dashed line), followed by a linear rise, with higher k values consistently increasing overall times for all n.

stances are solved faster than those with lower k. This range tends to occur near the peak runtime for the lower k, likely because at those m/n values, the higher k instances are nearly always embeddable—possibly even trivially so.

4. After the peak, as m/n continues to increase, instances become predominantly non-embeddable. In this regime, higher k instances consistently take longer to solve. This is likely due to the increased number of SAT variables and constraints introduced at higher k, which makes proving unsatisfiability more computationally demanding.

4 Discussion and Conclusion

To recap, this study investigates the *kUBE* problems for DAGs (NP-complete for $k \geq 2$), by analyzing embeddability across vertex count (n), edge count (m), and page count (k). Employing exhaustive enumeration of small DAGs, uniform sampling of larger instances, and SAT-based verification, we derive useful insights into the structural and computational properties of *kUBE*. We observe a consistent decline in embeddability as graph density (m/n) increases. For small DAGs ($4 \leq n \leq 6$), exhaustive enumeration reveals a smooth transition from fully embeddable at low density to entirely non-embeddable at high density. This pattern persists in larger DAGs ($n \leq 20$), where uniform sampling confirms that sparse graphs are universally embeddable, while dense graphs are predominantly not, highlighting the difficulty of embedding dense structures within limited page constraints (Figures 2, 3). Meanwhile, increasing the number of pages ($1 \leq k \leq 6$) significantly improves embeddability across all values of n. Higher values of k consistently shift embeddability curves upward, reflecting greater flexibility in edge assignments and enabling previously non-embeddable DAGs to admit valid embeddings (Figures 2, 3).

The SAT-1 encoding and the Kissat solver efficiently verified embeddability across diverse datasets, including 1,277 North Graph DAGs and 45,000 sampled instances, managing large graphs ($n = 20$) despite peak computational complexities. The scalability of SAT solvers suggests their potential for addressing other NP-complete graph-theoretic problems. It is also noted that the case $k = 1$ does not exhibit a computational runtime peak, in contrast to the pronounced peaks observed for $k \geq 2$. Since the $k = 1$ case is solvable in poly-time, this contrast suggests a possible connection between runtime behavior and underlying computational complexity. That is, examining whether the presence or absence of runtime peaks in other parameterized problems could serve as an empirical indicator of complexity class between poly-time solvable and NP-hard cases.

While our findings may not reveal fundamentally new structural properties, they provide empirical confirmation of a known pattern of *kUBE* embeddability across varying graph densities and page counts. This reinforces the utility of SAT-based formulations (e.g., [14]) for systematically exploring the parameter space of hard combinatorial problems. Although upward book embeddings are primarily of theoretical interest, our SAT-based approach demonstrates practical scalability in evaluating embeddability over large datasets. This may support future optimization variants relevant to multilayer circuit design [7, 17] or constrained network visualization [21, 23], where related abstractions are sometimes employed. Our work also aligns with broader studies of layout models such as queue and stack layouts [8], which share similar structural constraints.

A closer examination of the embeddability phase transitions reveals a sharp decline in satisfiability from 100% to 0% at specific m/n thresholds—ranging from 0.865 for $k = 1$ to 5.098 for $k = 6$—which quantifies how increasing page count permits denser DAGs to remain embeddable (Figure 4). Notably, these thresholds closely align with observed runtime peaks for $k \geq 2$, indicating a tight coupling between structural transition points and computational difficulty. In contrast, the absence of a peak for $k = 1$ reflects its polynomial-time solvability and qualitatively different runtime behavior. These patterns suggest a potential diagnostic role for runtime peaks in identifying complexity class transitions. A full characterization of these phase boundaries, along with their implications for solver performance and DAG structure, is presented in the appendix.

Acknowledgements

We sincerely thank all the reviewers for their thoughtful and constructive feedback, which greatly helped improve the quality and presentation of this work. We also gratefully acknowledge the support of the Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), as well as the research grant provided by the Faculty of Science and Environmental Studies at Lakehead University.

References

[1] M. A. Bekos, G. Da Lozzo, F. Frati, M. Gronemann, T. Mchedlidze, and C. N. Raftopoulou. Recognizing DAGs with page-number 2 is NP-complete. *Theoretical Computer Science*, 946:113689, 2023.

[2] M. A. Bekos, M. Kaufmann, and C. Zielke. The book embedding problem from a SAT-solving perspective. In *Graph Drawing and Network Visualization: 23rd International Symposium, GD 2015, Los Angeles, CA, USA, September 24-26, 2015, Revised Selected Papers 23*, pages 125–138. Springer, 2015.

[3] F. Bernhart and P. C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

[4] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt. CaDiCal, Gimsatul, IsaSAT and Kissat entering the SAT competition 2024. *SAT Competition 2024*, page 8, 2024.

[5] C. Binucci, G. Da Lozzo, E. Di Giacomo, W. Didimo, T. Mchedlidze, and M. Patrignani. Upward book embeddability of st-graphs: Complexity and algorithms. *Algorithmica*, 85(12):3521–3571, 2023.

[6] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the 12th IJCAI*, pages 331–337, 1991.

[7] F. R. Chung, F. T. Leighton, and A. L. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987.

[8] V. Dujmović and D. R. Wood. Stacks, queues and tracks: Layouts of graph subdivisions. *Discrete Mathematics & Theoretical Computer Science*, 7, 2005.

[9] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[10] J. K. Fichte, M. Hecher, and S. Szeider. A time leap challenge for SAT-solving. In *International Conference on Principles and Practice of Constraint Programming*, pages 267–285. Springer, 2020.

[11] N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda. SAT competition 2020. *Artificial Intelligence*, 301:103572, 2021.

[12] Graph Drawing Community. The International Symposium on Graph Drawing and Network Visualization. Accessed: 2025-02-20.

[13] L. S. Heath and S. V. Pemmaraju. Stack and queue layouts of directed acyclic graphs: Part ii. *SIAM Journal on Computing*, 28(5):1588–1626, 1999.

[14] M. J. H. Heule and S. Szeider. A SAT approach to clique-width. *ACM Transactions on Computational Logic*, 16(3):24:1–24:27, 2015.

[15] R. Kakimov and X. Tan. SAT for upward book embedding: An empirical study. In *Proceedings of the 38th Canadian Conference on Artificial Intelligence*. Canadian Artificial Intelligence Association, 2025.

[16] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[17] A. Knippel and F. Lardeux. The multi-layered network design problem. *European Journal of Operational Research*, 245(1):1–12, 2015.

[18] M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 13(1–2):15–20, 1967.

[19] J. Kuipers and G. Moffa. Uniform random generation of large acyclic digraphs. *Statistics and Computing*, 25:227–242, 2015.

[20] J. Kuipers and G. Moffa. *unifDAG: Uniform sampling of directed acyclic graphs*, 2022. R package version 1.0.4.

[21] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud. The state of the art in multilayer network visualization. *Computer Graphics Forum*, 38(6):125–149, 2019.

[22] B. Selman and H. J. Levesque. The hard and easy distribution of SAT problems. In *Proceedings of the 10th AAAI*, pages 440–446, 1992.

[23] R. Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

[24] The On-Line Encyclopedia of Integer Sequences. Sequence A003024: Number of directed acyclic graphs (DAGs) with n labeled nodes, 2024. Accessed: 2024-03-20.

[25] A. Wigderson. The complexity of the Hamiltonian circuit problem for maximal planar graphs. *EECS Department Report*, 298, 1982.

Appendix

The appendix provides supplementary details and analyses that support and extend the main findings of our study on the k-page upward book embedding problem ($kUBE$) for directed DAGs. It includes in Section A a detailed breakdown of the number of DAGs for small vertex counts ($n \leq 6$) across varying edge counts, as enumerated using Algorithm 1, and in Section B an in-depth exploration of phase transitions in embeddability. These transitions reveal critical thresholds in graph density (m/n) where embeddability shifts sharply, offering insights into the structural and computational complexities of $kUBE$.

A Numbers of DAGs with $n <= 6$

According to the On-Line Encyclopedia of Integer Sequences ([24, A003024]), the total number of DAGs is 25 for $n = 3$, 543 for $n = 4$, 29,281 for $n = 5$, and 3,781,503 for $n = 6$. In fact, we employ Algorithm 1 to determine their detailed breakdown across different edge counts (m), with the results presented in Tables 2, 3 and 4.

Table 2: Number of DAGs with (a) $n = 3$, and (b) $n = 4$ nodes, for different edge counts

(a) $n = 3$

Edges	0	1	2	3
# of DAGs	**1**	**6**	**12**	**6**

(b) $n = 4$

Edges	0	1	2	3	4	5	6
# of DAGs	**1**	**12**	**60**	**152**	**186**	**108**	**24**

Table 3: Number of DAGs with $n = 5$ nodes for different edge counts

Edges	0	1	2	3	4	5	6	7	8	9	10
# of DAGs	**1**	**20**	**180**	**940**	**3050**	**6180**	**7960**	**6540**	**3330**	**960**	**120**

Table 4: Number of DAGs with $n = 6$ nodes for different edge counts

Edges	0	1	2	3	4	5	6	7
# of DAGs	**1**	**30**	**420**	**3600**	**20790**	**83952**	**240480**	**496680**
Edges	8	9	10	11	12	13	14	15
# of DAGs	**750810**	**838130**	**691020**	**416160**	**178230**	**51480**	**9000**	**720**

B Phase Transition

From our study of DAGs—whether enumerating smaller instances or analyzing larger sampled ones with $n = 10$, 15, and 20—a clear phenomenon emerges: the presence of a phase transition in embeddability. In simple terms, a phase transition refers to a sudden shift in a system's behavior as a key parameter changes, much like water turning to ice as temperature drops; here, it manifests as a rapid drop from nearly all DAGs being embeddable to nearly none as the edge count m crosses a critical threshold.

To investigate this, one examines how a property (like embeddability) changes with a control parameter (such as m/n, or graph density), looking for a sharp transition rather than a gradual one, often marked by a specific point where the outcome flips dramatically, for example, where embeddability falls from 100% to near 0%. Observing this behavior in our previous results, we now delve into a detailed study of the phase transition, specifically using graph density (m/n) as the control variable, to characterize how embeddability drops sharply as m/n increases, indicating a rapid shift similar to a phase transition, while noting that the mean runtime peaks around this steep decline, across varying n and k.

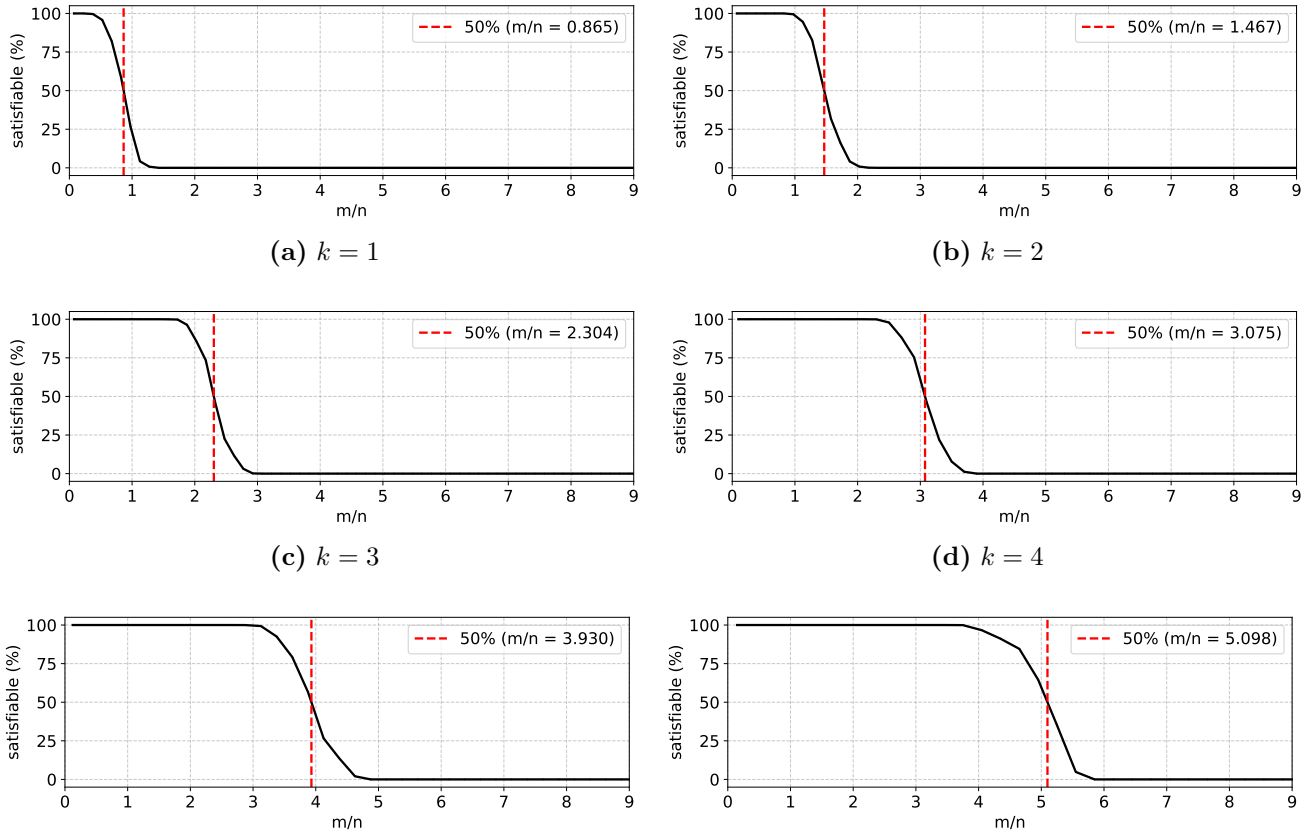
We used the 1,277 DAGs from the North Graph dataset [12] along with 45,000 DAGs generated through the sampling methods described in previous sections, analyzing their distribution in increasing graph density from $k = 1$ to 6 in terms of embeddability (Figures 4 a–f) and runtime (Figures 5 a–f).

Figure 4 in particular shows the percentage of embeddable DAGs decreasing rapidly at critical m/n values: 0.865 for $k = 1$, 1.467 for $k = 2$, 2.304 for $k = 3$, 3.075 for $k = 4$, 3.930 for $k = 5$, and 5.098 for $k = 6$, as marked by the 50% embeddability thresholds. These thresholds reveal a clear relationship: the critical m/n at which the phase transition occurs increases near-linearly with k, suggesting that each additional page allows the DAG to sustain a higher density before embeddability collapses, a trend we explore further to quantify its implications across varying n.

Figure 5 illustrates runtime behavior across $k = 1$ to 6 in subfigures (a–f), where each subfigure employs a scatter plot to depict runtime (in seconds, on a logarithmic scale) versus m/n for DAGs with $n = 7$ to 20, using distinct colors to differentiate node sizes and highlight trends across graph scales. For $k = 2$ to 6, the scatter plots reveal a pronounced peak in runtime that aligns closely with the critical m/n values from Figure 4—1.467 for $k = 2$, 2.304 for $k = 3$, 3.075 for $k = 4$, 3.930 for $k = 5$, and 5.098 for $k = 6$—where the phase transition occurs, underscoring the solver's peak complexity during the embeddability shift. Notably, the right side of each peak exhibits higher runtimes, as the increasing m/n corresponds to a larger number of edges in the graph, thereby requiring more time to verify embeddability.

The runtime behavior for $k = 1$ in Figure 5 differs markedly from cases where $k \geq 2$, due to differences in computational complexity. For $k \geq 2$, the NP-completeness of kUBE leads to runtime peaks at the phase transition (e.g., $m/n = 1.467$ for $k = 2$, increasing to 5.098 for $k = 6$), as shown in subfigures (b–f). In contrast, for $k = 1$, its polynomial-time solvability results in a gradual runtime increase without a peak at $m/n = 0.865$ in subfigure (a). This contrast underscores that when only one page is involved, a simpler decision process suffices, avoiding the exponential complexity spike that arises with multiple pages.

This behavior mirrors the distinction between 2SAT and 3SAT. While 3SAT is NP-complete [16] and exhibits a pronounced computational peak near its phase transition [6,22], 2SAT is solvable in linear time [9, 18] (e.g., via the implication graph and strongly connected components). As the clause-to-variable ratio in 2SAT grows, contradictions emerge more quickly, allowing for early solver termination. The lack of a runtime peak for the $k = 1$ case similarly reflects its polynomial solvability: contradictions or valid embeddings become apparent and are resolved efficiently in denser instances.

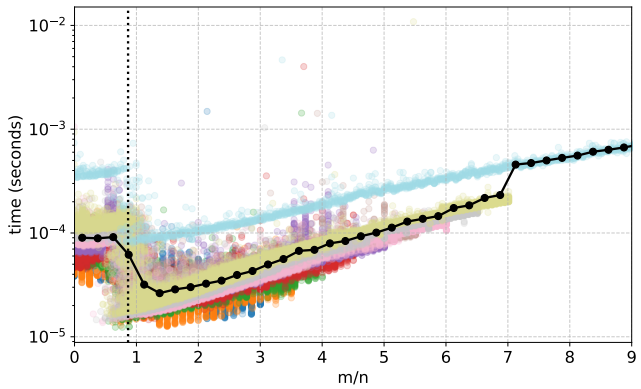


(a) $k = 1$ **(b)** $k = 2$

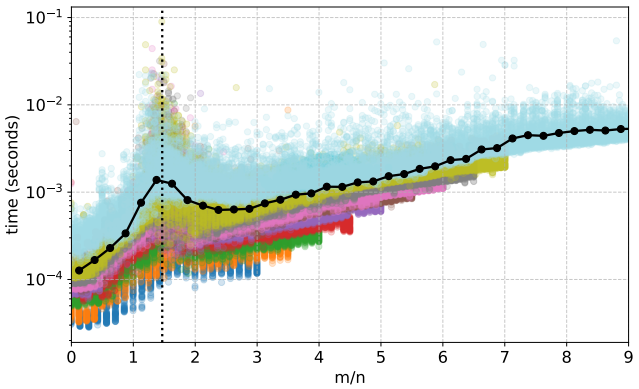
(c) $k = 3$ **(d)** $k = 4$

(e) $k = 5$ **(f)** $k = 6$

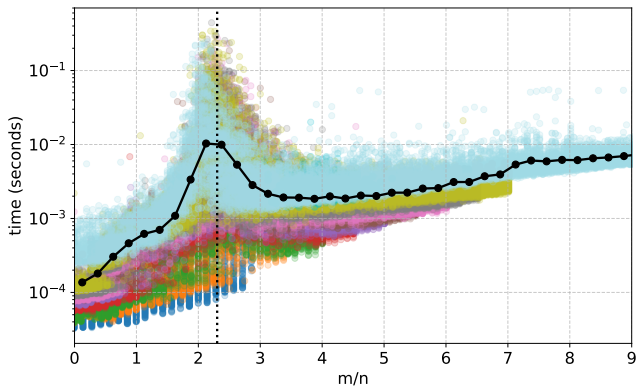
Figure 4: Percentage of DAGs embeddable in a k-page upward book embedding as a function of graph density (m/n). Vertical, red and dashed lines mark the 50% embeddability thresholds.



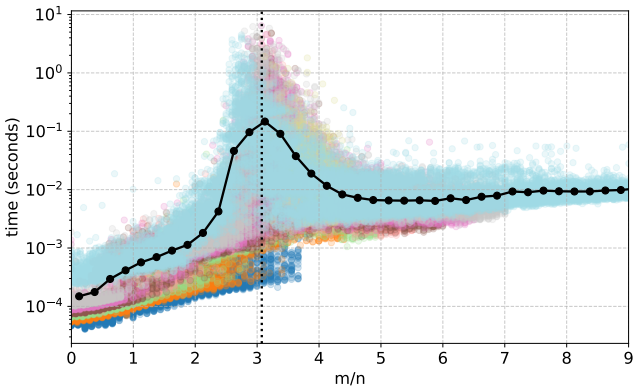
(a) k = 1



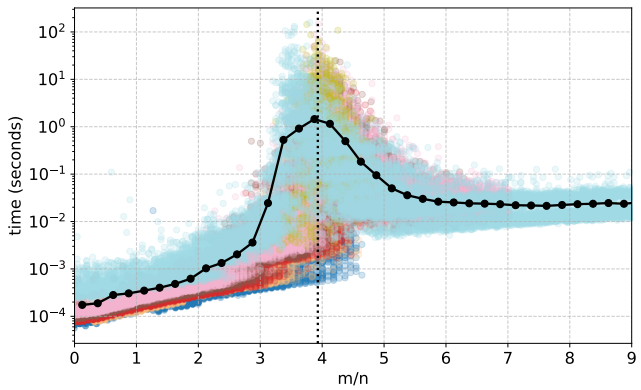
(b) k = 2



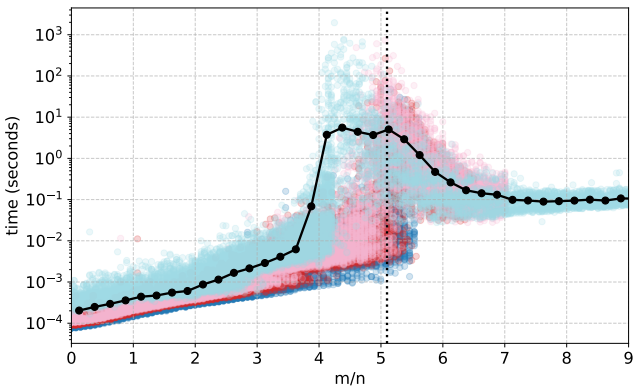
(c) k = 3



(d) k = 4



(e) k = 5



(f) k = 6

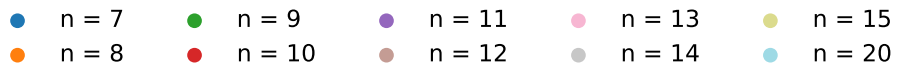


Figure 5: Run-time versus m/n for $k \in \{1, 2, \ldots, 6\}$. Each subplot presents a scatter plot of all combined instances for a fixed k, with points colored according to the value of n. The black line denotes the mean run-time per bin of m/n, and the vertical dotted line indicates the 50% embedding threshold.