



Graph Pattern Mining at Scale: Algorithms, Systems, and Unresolved Scalability Limits

Rohan Peralta^{1*}, Teresa Natarajan², Meera Khoury³

¹Department of Computer Science, Vistula Institute for Advanced Computer Science, New Zealand

²Department of Computer Science, Rhine Research Institute of Computer Science, Chile

³Department of Computer Science, Volta Institute of Computer Science, Jamaica

Article Info

Volume 1, Issue 1, 2026

doi: [10.62715/ijdwdm.2026.v1.i1.pp8-14](https://doi.org/10.62715/ijdwdm.2026.v1.i1.pp8-14)

Received: 30 September 2025

Accepted: 12 December 2025

Published: 11 February 2026

Abstract

Graph pattern mining asks which small structures occur in a graph, how often, and where. The family of problems includes frequent subgraph mining, motif and graphlet counting, clique listing and general subgraph matching, and all of them inherit the hardness of subgraph isomorphism. This narrative review follows the field from its algorithmic origins to the specialised systems of recent years. It first separates the problem variants and the support measures that make frequency well defined in a single large graph. It then describes the level-wise and pattern-growth algorithms developed for collections of small graphs, the constraint-based methods for single graphs, and the connection between pattern matching and worst-case optimal joins. The systems literature is organised into three generations: adaptations of vertex-centric frameworks, embedding-centric engines that enumerate and filter subgraphs, and pattern-aware engines that compile a matching plan for each pattern. The final part examines limits that engineering has not removed: output sizes that grow combinatorially with pattern size, sensitivity to degree skew, the cost of symmetry breaking for larger patterns, the difficulty of distributing an irregular search, and the weak comparability of published evaluations. Approximate counting and incremental mining are discussed as the most credible ways around these limits.

Keywords: *Graph pattern mining, Frequent subgraph mining, Subgraph isomorphism, Motif counting, Graph mining systems, Scalability*

© 2026 Rohan Peralta et al. This is an open access article under the CC BY license (<https://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

1. Introduction

Many questions asked of graph data are questions about small structures. A biologist wants to know which wiring patterns recur in a regulatory network more often than chance would predict (Milo et al., 2002). A chemist wants the substructures shared by a set of active compounds. An analyst of a payment network wants every instance of a suspicious cycle. Each task requires finding the occurrences, usually called embeddings or matches, of one or more pattern graphs inside a data graph, and the tasks differ mainly in whether the patterns are given in advance or must be discovered.

* Corresponding author: Rohan Peralta, Department of Computer Science, Vistula Institute for Advanced Computer Science, New Zealand.
E-mail: rohan.peralta@vistula.edu

© 2026 Rohan Peralta et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

These problems are grouped under the name graph pattern mining. They are expensive for a reason that no implementation can remove: deciding whether one graph contains another as a subgraph is NP-complete (Garey and Johnson, 1979), and the number of matches of even a small pattern can exceed the size of the data graph by many orders of magnitude. Progress has therefore come from pruning the search, avoiding repeated work and using hardware well, not from better worst-case bounds.

This review traces that progress in two parts, algorithms and systems, and then sets out the scalability limits that remain. Its purpose is to help a reader judge what current tools can be expected to do and where a new workload is likely to fail.

2. Scope and Approach of the Review

The article is a narrative review of the published literature on exact and approximate mining of subgraph patterns in static graphs. It concentrates on work that introduced an idea later adopted widely, and it relies on earlier surveys for breadth: Jiang et al. (2013) cover frequent subgraph mining algorithms in detail, and Ribeiro et al. (2021) cover subgraph counting. No systematic search was performed and no performance figures are reproduced, since results reported on different hardware, graphs and patterns cannot be set side by side with any confidence. Graph neural approaches to pattern counting and the mining of temporal graphs fall outside the scope.

3. Problem Variants and Their Complexity

Two settings must be kept apart. In the transactional setting the input is a collection of many small graphs, such as molecules, and a pattern is frequent if it occurs in at least a given number of them. Support counts graphs, not embeddings, so one occurrence per graph is enough. In the single-graph setting the input is one large graph and support must be defined over embeddings. Counting embeddings directly breaks the property on which all frequent pattern mining depends, namely that a pattern can be no more frequent than its subpatterns. Bringmann and Nijssen (2008) proposed the minimum image based support, which takes for each pattern vertex the number of distinct data vertices it maps to and then the minimum over pattern vertices. The measure is anti-monotone and cheap to compute, and it is now the usual choice.

A second distinction concerns what is asked for. Frequent subgraph mining discovers patterns. Subgraph matching lists the embeddings of a given pattern. Motif or graphlet counting returns the number of occurrences of every pattern up to a small size, and clique listing is the special case of complete patterns. A further choice is whether matches are induced, so that the data graph has no extra edges among the matched vertices, or merely edge-preserving. Systems differ in which of these they support natively, and a benchmark for one says little about another.

4. Algorithmic Foundations

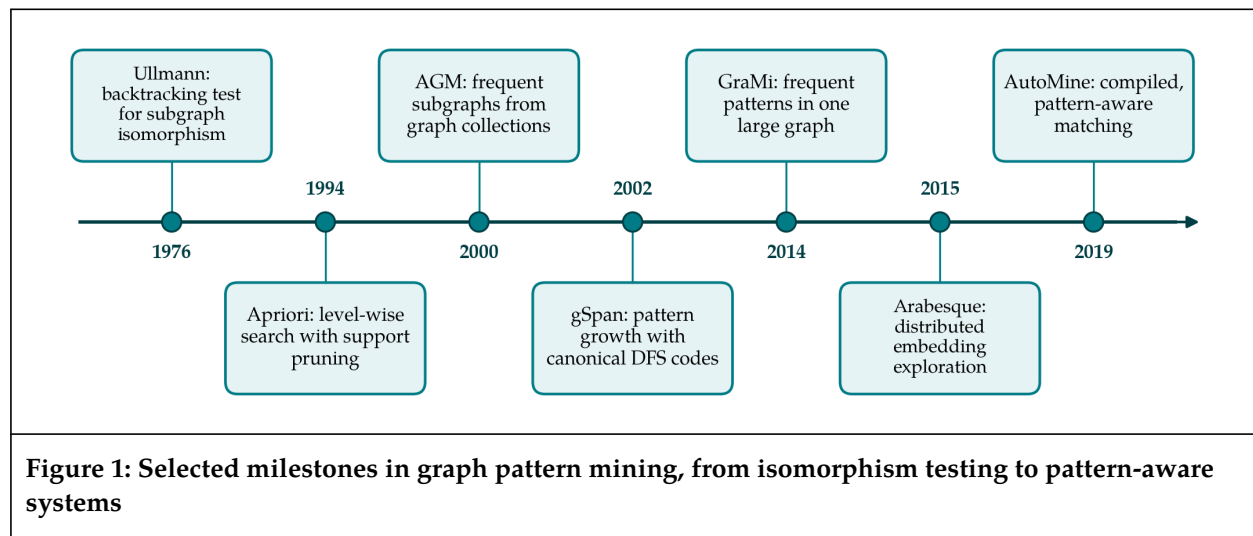
The algorithms on which present systems rest were developed over roughly four decades. Figure 1 marks the contributions that the remainder of this section and the next describe.

4.1. Matching a single pattern

The backtracking procedure of Ullmann (1976) extends a partial mapping one pattern vertex at a time and prunes candidates whose neighbourhoods cannot be matched. Almost every later matcher is a refinement of it. VF2 added feasibility rules that examine the frontier of the partial mapping and reduced memory use enough to handle graphs with thousands of vertices (Cordella et al., 2004). Later work concentrated on the order in which pattern vertices are matched and on filtering candidate sets before the search begins, since a poor order can make the same search slower by orders of magnitude.

A separate theoretical line treats a pattern as a join query in which each pattern edge is a relation. Ngo et al. (2018) gave join algorithms whose running time is bounded by the largest possible output for the query, a guarantee that conventional pairwise join plans do not meet; the triangle query is the standard example. These worst-case optimal joins extend one variable at a time by intersecting adjacency lists,

which is exactly what vertex-at-a-time backtracking does, and the result gave matching systems a principled account of why that strategy works.



4.2. Discovering frequent patterns in graph collections

Frequent subgraph mining borrowed its first design from itemset mining. Apriori generates candidate itemsets of size $k + 1$ from frequent ones of size k and discards any candidate with an infrequent subset (Agrawal and Srikant, 1994). AGM applied the scheme to graphs represented by adjacency matrices, growing patterns by a vertex at each level (Inokuchi et al., 2000), and FSG grew them by an edge (Kuramochi and Karypis, 2001). Both pay for candidate generation, which must join pairs of patterns and detect duplicates through canonical labelling, and for repeated isomorphism tests during support counting.

Pattern-growth methods removed candidate generation. gSpan assigns each pattern a canonical code derived from depth-first traversals and extends a pattern only along its rightmost path, so that the search space becomes a tree in which each pattern is visited once (Yan and Han, 2002). CloseGraph reports only closed patterns, those with no superpattern of equal support, and thereby shrinks the output considerably (Yan and Han, 2003). Gaston orders the search so that paths are enumerated first, then trees, then cyclic graphs, which defers the costly isomorphism work to the stage where it is unavoidable (Nijssen and Kok, 2004).

4.3. Mining a single large graph

Kuramochi and Karypis (2005) were among the first to mine frequent patterns in one sparse graph, using support measures based on sets of non-overlapping embeddings, which are themselves hard to compute. GraMi changed the formulation. It models the question of whether a pattern reaches the support threshold as a constraint satisfaction problem and stops searching as soon as enough distinct images have been found for every pattern vertex, so that the full set of embeddings is never enumerated (Elseidy et al., 2014). Avoiding enumeration wherever a decision or a count suffices has since become a general design principle.

4.4. Approximate counting

Where exact enumeration is out of reach, counts can be estimated. Colour coding assigns random colours to vertices and counts only matches whose vertices are all coloured differently, which can be done by dynamic programming for tree-like patterns (Alon et al., 1995). Sampling methods draw edges, paths or neighbourhoods and scale the counts observed. The survey by Ribeiro et al. (2021) classifies these estimators and notes that their accuracy guarantees weaken for rare patterns, which are often the patterns of interest.

5. Systems for Mining at Scale

General graph processing frameworks were not built for this workload. Pregel popularised the vertex-centric model, in which each vertex exchanges messages with its neighbours in synchronised rounds (Malewicz et al., 2010). The model suits computations whose state is proportional to the graph, such as PageRank. Pattern mining produces intermediate state proportional to the number of partial embeddings, and encoding those as messages overwhelms both memory and network. Dedicated systems followed, and Table 1 groups them into three generations.

Arabesque introduced the embedding as the unit of computation. The system enumerates all embeddings of size k , passes each to a user-defined filter and process function, and extends the survivors to size $k + 1$, distributing the work across machines and compressing the stored embeddings (Teixeira et al., 2015). The interface is simple and general, but the system explores every connected subgraph up to the target size whether or not it can match the pattern of interest, and it must test each embedding for canonicity to avoid counting it more than once. Fractal kept the model and replaced stored embeddings with depth-first recomputation from compact state, which trades processor time for memory (Dias et al., 2019). RStream took the opposite route on a single machine, expressing extension as a relational join over edge streams on disk (Wang et al., 2018).

Table 1: Three generations of graph pattern mining systems and their characteristic trade-offs

Generation	Core idea	Representative systems	Characteristic limit
Vertex-centric adaptation	Encode partial matches as messages between vertices	Pregel-style frameworks (Malewicz et al., 2010)	Message volume grows with embeddings
Embedding-centric, distributed	Enumerate embeddings level by level; filter with user code	Arabesque (Teixeira et al., 2015); Fractal (Dias et al., 2019)	Stores or recomputes many embeddings; canonicity checks
Embedding-centric, out-of-core	Express extension as relational joins streamed from disk	RStream (Wang et al., 2018)	Disk volume of intermediate tuples
Embedding-centric, accelerated	Extend and filter embeddings in bulk on CPU and GPU	Pangolin (Chen et al., 2020)	Device memory bounds embedding lists
Pattern-aware, compiled	Generate nested set-intersection loops for each pattern	AutoMine (Mawhirter and Wu, 2019); GraphPi (Shi et al., 2020)	Plan quality; many patterns to compile
Pattern-aware, interpreted	Derive matching order and symmetry breaking from the pattern	Peregrine (Jamshidi et al., 2020)	Assumes the graph fits in memory

The second shift was to let the pattern guide the search. AutoMine compiles each pattern into nested loops over set intersections and differences of adjacency lists, which is the worst-case optimal join strategy made concrete, and schedules the loops to share work between patterns (Mawhirter and Wu, 2019). Peregrine analyses the pattern at run time to obtain a matching order and a set of ordering constraints on the matched vertices; these constraints break the pattern's symmetries so that each match is produced once and no canonicity test is needed (Jamshidi et al., 2020). GraphPi searches over combinations of matching order and symmetry-breaking restrictions with a cost model and uses inclusion-exclusion to count without enumerating the innermost loops (Shi et al., 2020). Pangolin showed that the extend and filter steps can be run in bulk on graphics processors when embedding lists fit in device memory (Chen et al., 2020).

6. Unresolved Scalability Limits

Pattern-aware systems made small patterns on large graphs routine. They did not change the shape of the

problem, and five limits persist in the literature.

6.1. Output size and pattern size

The number of embeddings of a pattern with k vertices can grow as a polynomial of degree k in the size of the graph. Every exact enumerator is bound by that output. Published evaluations consequently stay with patterns of a handful of vertices, and each additional vertex typically multiplies the running time on a skewed graph. Counting techniques that avoid the innermost enumeration push the boundary outward by a level or so; they do not remove it. For frequent subgraph mining the difficulty is compounded, because the number of candidate patterns also grows super-exponentially with size and each requires its own support computation.

6.2. Degree skew

Real graphs have a few vertices of very high degree. Set intersections involving their adjacency lists dominate the running time, and the work attached to one such vertex can exceed that of millions of ordinary ones. Orienting edges from low to high degree helps for cliques and for some other symmetric patterns. For general patterns the remedy is less clear, and skew also undermines parallel scaling, since a static division of start vertices leaves most workers idle while a few finish the heavy tasks. Work stealing and fine-grained task splitting reduce the imbalance at the price of coordination overhead.

6.3. Symmetry and duplicate elimination

A pattern with many automorphisms is found many times unless the search is restricted. Ordering constraints solve this for small patterns, but the choice of constraints interacts with the matching order, the best combination depends on the data graph, and searching for it grows costly as the pattern gets larger. For discovery tasks the system must also decide whether two patterns encountered along different paths are the same, which is graph canonisation, a problem with no known polynomial-time algorithm for general graphs.

6.4. Distribution

Pattern search follows edges unpredictably, so a partitioned graph forces either remote adjacency lookups or replication of neighbourhoods. Systems that replicate the whole graph on every machine avoid the lookups and scale only until the graph outgrows one machine's memory. Systems that partition must ship partial embeddings or fetch adjacency lists on demand, and both make communication the bottleneck. No published design handles a graph larger than a single node's memory with the efficiency that the in-memory, pattern-aware engines achieve on smaller inputs.

6.5. Comparability of evaluations

Systems are evaluated on overlapping but different sets of graphs and patterns, with induced or non-induced semantics, with or without labels, and sometimes counting what a competitor enumerates. A reported speed-up may therefore reflect a different problem as much as a better engine. The field lacks a workload definition of the kind that decision-support benchmarks gave to relational systems, one that fixes the semantics, covers labelled and unlabelled graphs and includes patterns chosen to stress skew and symmetry.

7. Directions for Further Work

Three directions seem the most promising responses to these limits. The first is approximation with usable guarantees: estimators whose error can be bounded for the rare patterns that applications care about, and systems that expose the trade between accuracy and cost to the user. The second is incremental mining. Most production graphs change continuously, and recomputing pattern counts after each batch of edge updates wastes the previous result; restricting the search to matches that involve a changed edge is conceptually simple but complicates symmetry breaking and support maintenance. The third is tighter integration with query processing. Since pattern matching is a join, optimisers that combine worst-case optimal and conventional join operators, and that reason about labels and predicates through statistics,

could bring to mining the cost-based planning it currently lacks.

8. Conclusion

Graph pattern mining has advanced through a steady accumulation of pruning ideas: canonical codes that visit each pattern once, support measures that permit early termination, matching orders derived from the pattern, and symmetry breaking that eliminates duplicate matches before they are generated. Systems built on these ideas handle small patterns over very large graphs on a single well-provisioned machine. The limits that remain are structural. They concern output size, skew, symmetry in larger patterns, distribution and the comparability of results, and they are more likely to yield to approximation, incremental computation and better planning than to further tuning of exact enumeration.

Conflict of interest

To be declared by the author(s) before publication.

Funding

To be declared by the author(s) before publication.

References

- Agrawal, R. and Srikant, R. (1994). [Fast algorithms for mining association rules. *Proceedings of the 20th International Conference on Very Large Data Bases*, 487-499.](#)
- Alon, N., Yuster, R. and Zwick, U. (1995). [Color-coding. *Journal of the ACM*, 42\(4\): 844-856.](#)
<https://doi.org/10.1145/210332.210337>
- Bringmann, B. and Nijssen, S. (2008). [What is frequent in a single graph? *Advances in Knowledge Discovery and Data Mining \(PAKDD 2008\), Lecture Notes in Computer Science 5012*, 858-863.](#)
- Chen, X., Dathathri, R., Gill, G. and Pingali, K. (2020). [Pangolin: An efficient and flexible graph mining system on CPU and GPU. *Proceedings of the VLDB Endowment*, 13\(8\): 1190-1205.](#)
- Cordella, L.P., Foggia, P., Sansone, C. and Vento, M. (2004). [A \(sub\)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26\(10\): 1367-1372.](#) <https://doi.org/10.1109/TPAMI.2004.75>
- Dias, V., Teixeira, C.H.C., Guedes, D., Meira, W. and Parthasarathy, S. (2019). [Fractal: A general-purpose graph pattern mining system. *Proceedings of the 2019 International Conference on Management of Data*, 1357-1374.](#)
- Elseidy, M., Abdelhamid, E., Skiadopoulou, S. and Kalnis, P. (2014). [GraMi: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7\(7\): 517-528.](#)
<https://doi.org/10.14778/2732286.2732289>
- Garey, M.R. and Johnson, D.S. (1979). [Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco.](#)
- Inokuchi, A., Washio, T. and Motoda, H. (2000). [An apriori-based algorithm for mining frequent substructures from graph data. *Principles of Data Mining and Knowledge Discovery \(PKDD 2000\), Lecture Notes in Computer Science 1910*, 13-23.](#) https://doi.org/10.1007/3-540-45372-5_2
- Jamshidi, K., Mahadasa, R. and Vora, K. (2020). [Peregrine: A pattern-aware graph mining system. *Proceedings of the Fifteenth European Conference on Computer Systems \(EuroSys 2020\)*.](#)
- Jiang, C., Coenen, F. and Zito, M. (2013). [A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28\(1\): 75-105.](#) <https://doi.org/10.1017/S0269888912000331>
- Kuramochi, M. and Karypis, G. (2001). [Frequent subgraph discovery. *Proceedings of the 2001 IEEE International Conference on Data Mining*, 313-320.](#) <https://doi.org/10.1109/ICDM.2001.989534>
- Kuramochi, M. and Karypis, G. (2005). [Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11\(3\): 243-271.](#) <https://doi.org/10.1007/s10618-005-0003-9>

- Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N. and Czajkowski, G. (2010). *Pregel: A system for large-scale graph processing. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 135-146. <https://doi.org/10.1145/1807167.1807184>
- Mawhirter, D. and Wu, B. (2019). *AutoMine: Harmonizing high-level abstraction and high performance for graph mining. Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 509-523.
- Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D. and Alon, U. (2002). *Network motifs: Simple building blocks of complex networks. Science*, 298(5594): 824-827. <https://doi.org/10.1126/science.298.5594.824>
- Ngo, H.Q., Porat, E., Ré, C. and Rudra, A. (2018). *Worst-case optimal join algorithms. Journal of the ACM*, 65(3): Article 16. <https://doi.org/10.1145/3180143>
- Nijssen, S. and Kok, J.N. (2004). *A quickstart in frequent structure mining can make a difference. Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 647-652.
- Ribeiro, P., Paredes, P., Silva, M.E.P., Aparicio, D. and Silva, F. (2021). *A survey on subgraph counting: Concepts, algorithms, and applications to network motifs and graphlets. ACM Computing Surveys*, 54(2): Article 28. <https://doi.org/10.1145/3433652>
- Shi, T., Zhai, M., Xu, Y. and Zhai, J. (2020). *GraphPi: High performance graph pattern matching through effective redundancy elimination. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC20)*.
- Teixeira, C.H.C., Fonseca, A.J., Serafini, M., Siganos, G., Zaki, M.J. and Aboulmaga, A. (2015). *Arabesque: A system for distributed graph mining. Proceedings of the 25th Symposium on Operating Systems Principles*, 425-440. <https://doi.org/10.1145/2815400.2815410>
- Ullmann, J.R. (1976). *An algorithm for subgraph isomorphism. Journal of the ACM*, 23(1): 31-42. <https://doi.org/10.1145/321921.321925>
- Wang, K., Zuo, Z., Thorpe, J., Nguyen, T.Q. and Xu, G.H. (2018). *RStream: Marrying relational algebra with streaming for efficient graph mining on a single machine. Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*, 763-782.
- Yan, X. and Han, J. (2002). *gSpan: Graph-based substructure pattern mining. Proceedings of the 2002 IEEE International Conference on Data Mining*, 721-724. <https://doi.org/10.1109/ICDM.2002.1184038>
- Yan, X. and Han, J. (2003). *CloseGraph: Mining closed frequent graph patterns. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 286-295. <https://doi.org/10.1145/956750.956784>