

Learning-Based Graph Partitioning and Incremental Formal Verification for Scalable Optimization of Large-Scale Digital Circuits

Elin Martens

Department of Computer Science, University of North Texas, Denton, TX, USA.
elinmartens89@unt.edu

Avni Chhabra

Department of Computer Science, University of Houston, Houston, TX, USA.
avniwork@uh.edu

Abstract

Large-scale digital circuits now routinely contain hundreds of millions of gates, making scalable optimization, synthesis, and verification central problems in electronic design automation. Graph partitioning has long served as a foundational technique for decomposing these designs into manageable subproblems, yet classical partitioning objectives do not always align with the semantic and temporal dependencies that dominate formal verification. This paper examines the integration of learning-based graph partitioning with incremental formal verification as a system-level strategy for scalable optimization of large-scale digital circuits. It argues that partitioning should be understood not merely as a preprocessing heuristic but as an architectural decision that shapes verification cost, design reproducibility, infrastructure demand, and organizational governance. The discussion connects classical partitioning theory, symbolic model checking, satisfiability-based verification, logic synthesis, and graph representation learning. Particular attention is given to incremental verification, where partitioned artifacts are reused across design revisions, and to the structural trade-offs between partition granularity, cross-boundary communication, and proof stability. The paper also considers deployment in cloud and heterogeneous compute environments, robustness under noisy or incomplete design data, fairness in access to verification infrastructure, and policy implications for safety-critical semiconductor supply chains. Cross-domain comparisons with distributed computing, database sharding, and cyber-physical systems illustrate how partitioning decisions cascade across technical and institutional layers. The conclusion proposes that scalable circuit verification requires a socio-technical architecture in which learned partitioning, incremental proof management, and governance frameworks co-evolve. By treating graph partitioning and formal verification as coupled infrastructure rather than isolated algorithms, the paper outlines a research agenda for sustainable, robust, and equitable large-scale digital design.

Keywords

graph partitioning, formal verification, incremental verification, logic synthesis, large-scale circuits, machine learning, design automation, socio-technical infrastructure.

1. Introduction

The scale and complexity of modern digital circuits have outpaced many traditional assumptions in computer-aided design. A single system-on-chip may integrate heterogeneous

processing cores, memory hierarchies, interconnect fabrics, accelerators, and security modules, while its verification environment must reason about concurrency, power states, and manufacturing variability. Graph partitioning has therefore remained a durable strategy because it converts an intractable global problem into a collection of smaller, partially independent problems [1]. Multilevel partitioning methods, in particular, have demonstrated that high-quality decompositions can be computed for irregular graphs with millions of vertices [2]. These methods rely on coarsening, initial partitioning, and refinement, and they have been adapted to hypergraphs, directed graphs, and constrained partitioning problems [3]. Yet the rising cost of formal verification suggests that partitioning decisions cannot be evaluated only by cut size or load balance. They must also be assessed by their effects on proof complexity, counterexample interpretability, incremental reuse, and the human and computational infrastructure required to sustain verification over many design revisions [4].

Formal verification provides mathematical evidence that a design satisfies a specification, but its practical scalability depends on symbolic representations, abstraction, and decomposition. Symbolic model checking using binary decision diagrams introduced a powerful way to represent state sets, although it can suffer from representation blow-up in arithmetic and control-intensive designs [6]. Bounded model checking and satisfiability-based techniques shifted the burden toward efficient Boolean reasoning and have become central to industrial verification flows [5]. Boolean function manipulation through graph-based algorithms remains a conceptual foundation for many synthesis and verification operations [7]. These methods are not merely algorithmic alternatives; they define different cost profiles, failure modes, and opportunities for partitioning. A partitioning strategy that reduces the number of state variables in each subproblem may help symbolic methods, while a strategy that isolates control logic may assist satisfiability solvers. In both cases, the partition boundary becomes a site of information exchange, and the verification architecture must manage assumptions, invariants, and proof obligations across that boundary [11].

Classical graph partitioning heuristics such as Kernighan-Lin and Fiduccia-Mattheyses provided early local search mechanisms for improving bisections and multi-way partitions [8]. Subsequent work introduced multilevel and spectral approaches, recursive bisection, and approximation algorithms with formal guarantees under restricted conditions [12], [13]. These contributions remain essential, but they often optimize structural objectives that only indirectly correlate with verification effort. Recent research has begun to explore learning-based partitioning, including graph edge re-growth and partitioning for verification of large designs in logic synthesis [9]. Such work signals a broader shift: instead of relying solely on handcrafted objective functions, partitioning can be informed by learned representations of circuit structure, verification history, and solver behavior. This paper develops a system-level analysis of that shift, with emphasis on incremental formal verification, deployment architecture, robustness, fairness, governance, and sustainability.

2. Graph Partitioning as a Scalability Foundation for Digital Circuit Analysis

Graph partitioning is traditionally framed as a combinatorial optimization problem: given a graph, divide its vertices into a fixed number of parts while minimizing edge cut and balancing part sizes. For digital circuits, vertices may represent gates, nets, modules, or verification properties, and edges may represent electrical connectivity, logical dependency, or temporal influence. The choice of representation is consequential because it determines what kinds of locality can be preserved. A gate-level hypergraph captures fan-in and fan-out relationships that matter for logic synthesis and test generation, while a register-transfer-level

graph captures dataflow and control dependencies that matter for sequential verification. When partitioning is used for formal verification, the objective should reflect not only communication cost but also the semantic difficulty of reasoning across boundaries [1], [11]. A small cut that separates a datapath from its controller may still produce difficult proof obligations because the controller constrains when data is valid. Conversely, a larger cut that preserves a complete protocol state machine may simplify verification by keeping critical invariants local.

Multilevel partitioning has been successful because it exploits the hierarchical and clustered structure of large graphs [2]. Coarsening merges strongly connected vertices, initial partitioning finds a coarse decomposition, and refinement improves the boundary while respecting constraints. This approach is computationally efficient and often produces partitions that are qualitatively better than single-level heuristics. However, the hierarchy learned during coarsening is not necessarily aligned with the hierarchy of verification properties. A design may have a natural module hierarchy that differs from its connectivity clustering. If partitioning ignores specification structure, verification may be forced to reason across module boundaries that were intentionally abstracted. If partitioning respects only specification structure, it may create severe load imbalance or excessive cross-partition communication. The trade-off is therefore architectural rather than purely algorithmic [3]. System designers must decide which hierarchy is authoritative and how to reconcile competing views of the same circuit.

Classical local search methods such as Kernighan-Lin and Fiduccia-Mattheyses remain important because they clarify the role of incremental improvement and move evaluation [8], [10]. These methods are attractive when a partition already exists and must be adapted after small design changes. In incremental verification flows, this scenario is common: a late-stage bug fix or timing optimization may alter a small set of gates, yet the verification infrastructure must decide whether the existing partition remains valid or whether repartitioning is worth the cost. Local search can preserve stability by limiting movement, but it may also trap the flow in a poor decomposition. Learning-based methods can potentially predict which moves are likely to reduce future verification effort, rather than only reducing immediate cut size. The distinction matters because verification cost is path-dependent: a partition that looks suboptimal at time zero may be valuable later if it isolates frequently modified logic or stabilizes proof reuse [9].

Recursive bisection and spectral methods provide complementary perspectives. Recursive bisection repeatedly splits a graph into two parts, which can simplify balancing and produce hierarchical decompositions suitable for parallel verification [13]. Spectral methods use eigenvectors of graph Laplacians to identify global structure, but they can be expensive at extreme scale and may not handle hyperedges naturally. Approximation algorithms offer guarantees under specific graph classes, yet real circuits often violate assumptions of uniform degree, planarity, or bounded treewidth [12]. The practical lesson is that no single partitioning objective dominates. A verification-aware system should support multiple partition views, maintain metadata about why each view was chosen, and allow incremental switching when the design or specification changes. This is a governance problem as much as an algorithmic one, because different teams may own different views and may have different incentives to preserve or revise them [11].

3. Learning-Based Partitioning and Representation Learning for Circuit Graphs

Learning-based graph partitioning seeks to improve decomposition by training models on historical data, structural features, or solver feedback. A circuit graph can be represented through node features such as gate type, fan-in count, switching activity, and timing slack, as well as edge features such as net weight, logical implication, and verification relevance. Graph neural networks can aggregate neighborhood information and produce embeddings that capture local and global structure [20], [21]. These embeddings can then be used to score candidate cuts, predict partition quality, or guide refinement moves. The appeal is not that learning replaces combinatorial optimization, but that it can provide informed priors where classical objectives are incomplete. In verification, the relevant prior may include which partitions previously produced short proofs, which boundaries generated spurious counterexamples, and which subproblems were reused across revisions.

The required recent work on graph edge re-growth and partitioning for large designs in logic synthesis illustrates how structural regeneration can be combined with partitioning decisions [9]. The significance of this direction is systemic: it treats partitioning as a dynamic process that responds to edge growth, logic restructuring, and verification demands, rather than a one-time preprocessing step. Such dynamism is important because logic synthesis transforms the graph repeatedly. Optimization passes can add and remove nodes, rewrite subgraphs, and change connectivity without altering the high-level specification. A partition that is optimal before synthesis may become inefficient afterward. A learning-based system can monitor these transformations and recommend when to regrow edges, merge partitions, or split a partition to preserve verification locality. The goal is not perpetual repartitioning, which would destroy incremental reuse, but selective adaptation when the expected verification benefit exceeds the cost of reorganizing proofs and infrastructure.

Attention mechanisms and transformer-style architectures offer another route for modeling long-range dependencies in circuit graphs [22]. Unlike convolutional or message-passing models that emphasize local neighborhoods, attention can assign different weights to distant nodes, which may help identify control signals that influence many parts of the design. However, attention models are data-hungry and computationally expensive. Their deployment in electronic design automation must therefore be justified by clear gains in partition quality or verification time. A practical architecture might use lightweight graph embeddings for online decisions and heavier attention models for offline analysis of historical verification logs. This separation of timescales is common in large-scale systems: fast heuristics handle immediate choices, while slower learning components update policies based on accumulated evidence. The challenge is to ensure that offline models remain aligned with the current design, because circuit libraries, synthesis tools, and verification methodologies evolve over time [18].

Learning-based partitioning also raises questions about interpretability and accountability. If a model recommends a partition that later leads to a verification failure, engineers need to understand whether the failure is due to an incomplete specification, a solver limitation, or an unsuitable decomposition. Pure end-to-end learning may obscure these distinctions. A more robust approach combines learned scores with explicit structural diagnostics, such as boundary complexity, fan-in cone depth, and state-variable overlap. These diagnostics can be exposed to verification engineers and used in governance reviews. In safety-critical domains, the ability to justify a partitioning decision may be as important as the decision itself. The system should therefore maintain provenance records linking partition choices to training data,

design revisions, and verification outcomes. This kind of provenance is analogous to data lineage in distributed data systems and is essential for reproducible engineering [4], [15].

4. Incremental Formal Verification in Partitioned Design Flows

Incremental formal verification aims to reuse previous verification results when a design changes. If a property was proven on an earlier revision, and the change does not affect the cone of influence of that property, the proof can be retained. If the change affects only a small region, local re-verification may suffice. Partitioning can support incrementality by creating stable verification units with explicit interfaces. Each partition can have a contract describing assumptions about its inputs and guarantees about its outputs. When a partition changes, only its contract needs to be rechecked, provided that the surrounding assumptions remain valid. This contract-based approach is familiar in software verification and hardware intellectual property integration, but it becomes more complex in large-scale circuits because assumptions may be implicit in timing, power, and test modes [4], [11]. A partition boundary that ignores clock domain crossings or reset sequencing may produce contracts that are technically valid but operationally misleading.

Satisfiability-based verification and bounded model checking are particularly sensitive to partitioning. A satisfiability solver benefits from small, well-structured formulas with limited variable interaction. Partitioning can reduce formula size by isolating independent subproblems, but it can also introduce interface variables that increase coupling. If the partition boundary cuts through a datapath, the solver may need to reason about partial arithmetic functions that are harder than the original problem. If the boundary aligns with control-flow branches, the solver may decompose naturally into cases. Learning-based partitioning can help by predicting which boundaries are likely to produce easy formulas, using features from previous solver runs. However, the system must also manage the incremental proof database. Unsat cores, interpolants, and proof traces are valuable artifacts, but they are specific to a particular partition and solver configuration. Reusing them across repartitioned designs is non-trivial [5].

Incremental verification also has an organizational dimension. Different teams may own different partitions, and a change in one partition can invalidate assumptions in another. Without clear ownership and notification mechanisms, verification results can become stale or contradictory. A partitioned verification infrastructure should therefore include a dependency graph that links properties, partitions, contracts, and design revisions. When a partition is modified, the system can identify affected properties and schedule re-verification. This dependency graph is itself a graph partitioning problem, but its edges represent proof dependencies rather than electrical connections. Learning-based methods can help prioritize which invalidated proofs to recheck first, especially when compute resources are limited. The governance challenge is to ensure that priorities reflect both technical risk and organizational responsibility [15].

The tension between stability and optimality is central to incremental partitioned verification. Repartitioning can improve solver performance, but it can also discard accumulated proof artifacts and disrupt team workflows. A conservative strategy preserves partitions unless the expected gain is large. An aggressive strategy repartitions frequently to track design changes. The right balance depends on the verification phase. Early in design, frequent repartitioning may be acceptable because proofs are less mature and the design is still fluid. Late in design, stability is more valuable because any change can trigger expensive re-verification and schedule risk. A learning-based controller can estimate the phase from design activity, proof

maturity, and schedule metadata, then choose a repartitioning policy accordingly. This is a form of adaptive infrastructure management, and it requires close integration between design tools, verification engines, and project governance [9], [11].

5. System Architecture, Deployment, and Infrastructure Considerations

Deploying learning-based partitioning and incremental verification at scale requires an architecture that separates concerns while maintaining feedback loops. At the lowest layer, design databases and synthesis engines produce circuit graphs and revision histories. Above that, partitioning services compute and store decompositions, metadata, and contracts. Verification engines consume partitions and return proof results, solver statistics, and counterexamples. A learning layer ingests these signals to update models and policies. Finally, a governance layer defines access control, audit requirements, and resource quotas. This layered architecture resembles modern data platforms, where storage, computation, orchestration, and governance are distinct but interoperable [23]. In electronic design automation, however, the data is highly structured and the cost of recomputation is high, so caching and provenance are critical. A partition should be treated as a versioned artifact with dependencies, not as a transient directory.

Compute infrastructure for this architecture may span on-premises clusters, private clouds, and public clouds. Formal verification is often bursty: a single regression can demand thousands of solver instances, then subside. Learning-based partitioning adds training and inference workloads, which may have different hardware preferences. Graph neural networks benefit from accelerators with high memory bandwidth, while satisfiability solvers may benefit from high single-thread performance and large memory. The deployment strategy must therefore schedule heterogeneous workloads without starving verification jobs. Interconnect topology also matters. If partitions are distributed across nodes, communication of proof artifacts and counterexamples can dominate runtime. This is analogous to distributed graph processing, where partitioning quality directly affects network traffic [23]. A system-level design should co-locate dependent partitions when possible and replicate shared metadata for fast access.

Robustness is a major concern in such infrastructure. Design data may be incomplete, inconsistent, or noisy due to tool bugs, library mismatches, or manual overrides. A learning model trained on clean historical data may produce poor partitions when confronted with anomalous revisions. The system should detect distribution shift by monitoring features such as graph density, fan-out distribution, and solver timeout rates. If shift is detected, the controller can fall back to classical partitioning or conservative incremental reuse. Robustness also requires graceful degradation. If a learning service is unavailable, verification should continue with rule-based partitioning. If a partition store is corrupted, the system should reconstruct from design databases and proof logs where possible. These requirements are not unique to circuit design; they are standard concerns in critical infrastructure, but they are often underemphasized in algorithm-centric research [18].

6. Robustness, Fairness, Governance, and Policy Implications

The integration of learning-based partitioning and formal verification raises fairness questions. Access to advanced verification infrastructure can determine which organizations can design complex chips, which teams can meet schedules, and which regions can participate in semiconductor innovation. If high-quality partitioning models are trained primarily on proprietary designs from a few large firms, the resulting tools may perform poorly on designs

from smaller organizations, academic groups, or emerging markets. This is a form of data inequality that can translate into design inequality. Public benchmarks and open verification datasets can help, but they must represent diverse design styles, technology nodes, and application domains [17]. Fairness also applies within organizations: if verification resources are allocated based on learned priorities, some teams may be systematically deprioritized. Governance should include transparency about how priorities are set and appeal mechanisms for contested decisions.

Policy implications extend beyond individual firms. Governments increasingly view semiconductors as strategic infrastructure, and verification is a bottleneck in supply chain resilience. If verification flows depend on opaque learned models or foreign cloud services, supply chain risks may increase. Policy responses might include support for open verification tooling, standards for proof provenance, and incentives for reproducible partitioning benchmarks. Export controls and data localization rules can also affect how design data and verification artifacts move across borders. A partitioned verification architecture that stores proof artifacts in multiple jurisdictions may face legal complexity. These issues are not merely external constraints; they shape the technical design of partitioning and incremental verification systems. For example, a system that must keep certain design data on-premises may need federated learning or local model adaptation [18]. Such requirements influence partition granularity, because smaller partitions may be easier to isolate for compliance.

Governance of learned partitioning also involves accountability for errors. If a verification failure escapes to silicon, investigators will ask whether the partitioning policy contributed to the miss. Was a property assigned to a partition where its assumptions were invalid? Was an incremental proof incorrectly reused? Was a learned model biased toward reducing runtime at the expense of coverage? Answering these questions requires audit trails that link model versions, partition versions, solver versions, and design revisions. This is similar to model governance in other safety-critical domains, where documentation and traceability are mandatory [4]. The technical community should develop standard formats for partition provenance and proof contracts. Without such standards, cross-organizational verification will remain fragile and difficult to automate.

7. Case Illustrations and Cross-Domain Comparisons

A useful illustration is a multi-core processor with a shared cache coherence protocol. Verification of coherence properties requires reasoning about interactions among cores, cache controllers, and interconnect. A connectivity-based partition might separate each core with its local cache, producing balanced parts. However, coherence invariants are global and depend on message ordering. A verification-aware partition might instead group the interconnect and directory with several cores, creating a larger but semantically coherent partition. The trade-off is clear: smaller partitions reduce local state space but increase cross-boundary assumptions. Incremental verification would then monitor changes in the coherence protocol and decide whether to reuse proofs for unaffected cores. Learning-based models could predict which protocol changes are likely to invalidate global invariants based on historical bug patterns [9], [11].

Another illustration comes from cryptographic accelerators, where datapath and control logic are tightly coupled and security properties must hold under adversarial inputs. Partitioning for formal verification must preserve the relationship between secret-dependent operations and observable outputs. A cut that separates key scheduling from encryption rounds may make verification easier locally but harder globally because security depends on the entire sequence.

In this setting, fairness and policy concerns are acute: verification failures can have security consequences, and proof provenance may be subject to export controls. The system architecture must support strong access control and auditability while still allowing incremental reuse. This resembles cross-domain challenges in distributed databases, where sharding improves scalability but complicates transactional guarantees [23]. The lesson is that partitioning always creates a boundary, and boundaries create both efficiency and risk.

Cross-domain comparison with cloud data systems highlights the importance of elastic infrastructure. Database sharding partitions data to improve throughput, but resharding is expensive and can introduce inconsistency. Similarly, repartitioning a circuit for verification can invalidate proofs and disrupt workflows. Both domains benefit from stable partition keys, incremental migration, and observability. Cross-domain comparison with cyber-physical systems highlights the role of timing and environment. A partition that ignores physical timing assumptions may verify a controller in isolation but fail when integrated with sensors and actuators. Formal verification of digital circuits is often more discrete, but timing, power, and analog interfaces introduce continuous concerns that partitioning can obscure. A robust architecture should therefore include interface contracts that cover not only logical values but also timing assumptions and environmental constraints [4], [11].

8. Sustainability and Long-Term Maintainability

Sustainability in large-scale circuit verification has at least three dimensions: computational energy, engineering effort, and institutional continuity. Learning-based partitioning can reduce computational energy if it shortens verification runs and avoids redundant proofs. However, training graph neural networks and running inference also consume energy. The net benefit depends on how often models are retrained, how many designs share the same model, and whether inference replaces or supplements classical partitioning. A lifecycle assessment should account for data collection, model training, deployment, and eventual decommissioning. In many cases, the largest energy cost is repeated solver execution, so even modest improvements in partition quality can yield significant savings. Incremental verification is particularly promising because it avoids recomputation rather than merely accelerating it [9], [18].

Engineering effort is another sustainability concern. Verification flows are often maintained by small teams with deep domain knowledge. Introducing learned components can increase maintenance burden if models drift, dependencies break, or explanations are unclear. A sustainable architecture should favor modularity, backward compatibility, and clear interfaces. Classical partitioning algorithms should remain available as fallbacks and as baselines for evaluating learned methods. Documentation should include not only how models are trained but also when they should not be used. This is a form of technical debt management. Just as software systems accumulate dependencies, verification systems accumulate proof artifacts, scripts, and assumptions. Partitioning metadata can help manage this debt by making dependencies explicit and enabling selective cleanup. Without such metadata, incremental verification can become a source of hidden fragility [15].

Institutional continuity matters because verification evidence may need to remain valid for years. A chip design may be revised, rebranded, or integrated into new systems. If the verification environment is tied to a specific vendor, cloud region, or model version, reproducing past results may become difficult. Open standards for proof contracts, partition provenance, and solver interfaces can improve continuity. They also support collaboration among universities, startups, and established firms. The policy implication is that public

investment in verification infrastructure can have long-term benefits beyond any single design. By treating partitioning and incremental verification as shared infrastructure, the semiconductor ecosystem can reduce duplication and improve resilience. This is analogous to investment in open-source software and public data repositories, where collective maintenance sustains individual innovation [17], [18].

9. Conclusion

This paper has argued that learning-based graph partitioning and incremental formal verification should be understood as coupled components of a socio-technical infrastructure for large-scale digital circuits. Classical partitioning provides essential foundations, but verification-aware partitioning must account for semantic dependencies, proof reuse, and organizational boundaries. Learning-based methods can improve decomposition by using historical verification data and structural representations, but they also introduce risks of opacity, data inequality, and maintenance burden. Incremental verification offers a path to scalability by reusing proofs across revisions, yet its benefits depend on stable partitions, explicit contracts, and robust provenance. The system architecture must therefore integrate design databases, partitioning services, verification engines, learning layers, and governance controls. Deployment across heterogeneous compute environments requires careful attention to scheduling, communication, and graceful degradation.

The broader implications extend to fairness, policy, and sustainability. Access to advanced verification infrastructure shapes who can participate in semiconductor innovation, and opaque learned models can create new dependencies and accountability gaps. Policy frameworks should encourage open benchmarks, reproducible proof provenance, and transparent governance of automated decisions. Sustainability requires reducing redundant computation while also maintaining the human and institutional capacity to understand and audit verification results. Cross-domain comparisons with distributed databases and cyber-physical systems reinforce the lesson that partitioning creates boundaries, and boundaries create both efficiency and risk. Future research should develop verification-aware graph representations, incremental proof management across repartitioning events, and governance models that align technical optimization with public interest. By treating partitioning and verification as infrastructure rather than isolated algorithms, the field can pursue scalable optimization without sacrificing robustness, fairness, or long-term maintainability.

References

1. Alpert, C. J., & Kahng, A. B. (1995). Recent directions in netlist partitioning: A survey. *Integration, the VLSI Journal*, 19(1-2), 1-81. [https://doi.org/10.1016/0167-9260\(95\)00008-4](https://doi.org/10.1016/0167-9260(95)00008-4)
2. Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 359-392. <https://doi.org/10.1137/S1064827595287997>
3. Hendrickson, B., & Leland, R. (1995). A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing* (pp. 28-es). ACM. <https://doi.org/10.1145/224170.224228>
4. Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model checking*. MIT Press.

5. Biere, A., Cimatti, A., Clarke, E. M., & Zhu, Y. (1999). Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 193-207). Springer. https://doi.org/10.1007/3-540-49059-0_14
6. McMillan, K. L. (1993). *Symbolic model checking*. Kluwer Academic Publishers.
7. Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 677-691. <https://doi.org/10.1109/TC.1986.1676819>
8. Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2), 291-307. <https://doi.org/10.1002/j.1538-7305.1970.tb01770.x>
9. Thorat, Kiran, et al. "GROOT: Graph edge re-growth and partitioning for the verification of large designs in logic synthesis." 2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD). IEEE, 2025.
10. Fiduccia, C. M., & Mattheyses, R. M. (1982). A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference* (pp. 175-181). IEEE. <https://doi.org/10.1145/800263.809204>
11. Kahng, A. B., Lienig, J., Markov, I. L., & Hu, J. (2011). *VLSI physical design: From graph partitioning to timing closure*. Springer. <https://doi.org/10.1007/978-90-481-9591-6>
12. Bui, T. N., Chaudhuri, S., Leighton, F. T., & Sipser, M. (1987). Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2), 171-191. <https://doi.org/10.1007/BF02579448>
13. Simon, H. D., & Teng, S.-H. (1997). How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5), 1436-1445. <https://doi.org/10.1137/S1064827593255135>
14. Brayton, R. K., Hachtel, G. D., McMullen, C. T., & Sangiovanni-Vincentelli, A. L. (1984). *Logic minimization algorithms for VLSI synthesis*. Kluwer Academic Publishers.
15. Sentovich, E. M., Singh, K. J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P. R., Brayton, R. K., & Sangiovanni-Vincentelli, A. (1992). *SIS: A system for sequential circuit synthesis*. University of California, Berkeley.
16. Mishchenko, A., Chatterjee, S., & Brayton, R. (2006). DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proceedings of the 43rd Design Automation Conference* (pp. 532-535). ACM. <https://doi.org/10.1145/1146909.1147048>
17. Amaru, L., Gaillardon, P.-E., & De Micheli, G. (2016). The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic and Synthesis*.
18. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. <https://doi.org/10.1038/nature14539>
19. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489. <https://doi.org/10.1038/nature16961>

20. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations.
21. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems (pp. 1024-1034).
22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998-6008).
23. Dally, W. J., & Towles, B. (2004). Principles and practices of interconnection networks. Morgan Kaufmann.