Efficient Parallel Verification of Natural Deduction Proof Graphs

James Oswald ● Brandon Rozek
Rensselaer AI and Reasoning (RAIR) Laboratory
Rensselaer Polytechnic Institute

Introduction and Objectives

We present a set of parallel algorithms for the formal verification of graph based natural deduction style proofs. We compare our approach with a single-threaded implementation on the AIMOS super computer and perform an ablation study with respect to possible optimizations and various proof topologies. We conclude parallization leads to an order of magnitude increase in verification performance on particular topologies of proofs.

Natural Deduction, Inference Rules, Proof Graphs

Natural Deduction is a logic calculi independently proposed in (Gentzen 1935, Jaśkowski 1934) in an effort to emulate human-level reasoning through assumptions and chains of inference. The rules of natural deduction associate each formula with a set of assumptions from which they are derived (Γ, Σ in the tables). The rules specify the syntactic and semantic conditions for the preceding formulae/assumptions and dictate when applied how the resulting formulae and assumptions are transformed.

Figure: Table 1: (Some) Introduction Rules

Rule Name	Condition(s)	Result
Conjunctive Introduction	$p\ (\Gamma), q\ (\Sigma)$	$p \wedge q\ (\Gamma \cup \Sigma)$
Disjunctive Introduction	$p\ (\Gamma)$	$p \vee q\ (\Gamma)$
Conditional Introduction	$q \vdash p\ (q \cup \Gamma)$	$p \rightarrow q\ (\Gamma)$
Negation Introduction	$p\ (q \cup \Gamma), \neg p\ (\Sigma)$	$\neg q\ (\Gamma \cup \Sigma)$

Figure: Table: (Some) Elimination Rules

Rule Name	Condition(s)	Result
Conjunctive Elimination	$p \wedge q\ (\Gamma)$	$p\ (\Gamma), q\ (\Gamma)$
Conditional Elimination	$p \rightarrow q(\Gamma), p\ (\Sigma)$	$q\ (\Gamma \cup \Sigma)$
Negation Elimination	$p\ (\neg q \cup \Gamma), \neg p\ (\Sigma)$	$q\ (\Gamma \cup \Sigma)$

These proofs can be represented as directed acyclic hypergraphs in which a labeled edge representing a rule connects the condition formulae to the result formulae. (Bringsjord, Govindarajulu, Taylor & Bringsjord 2022)

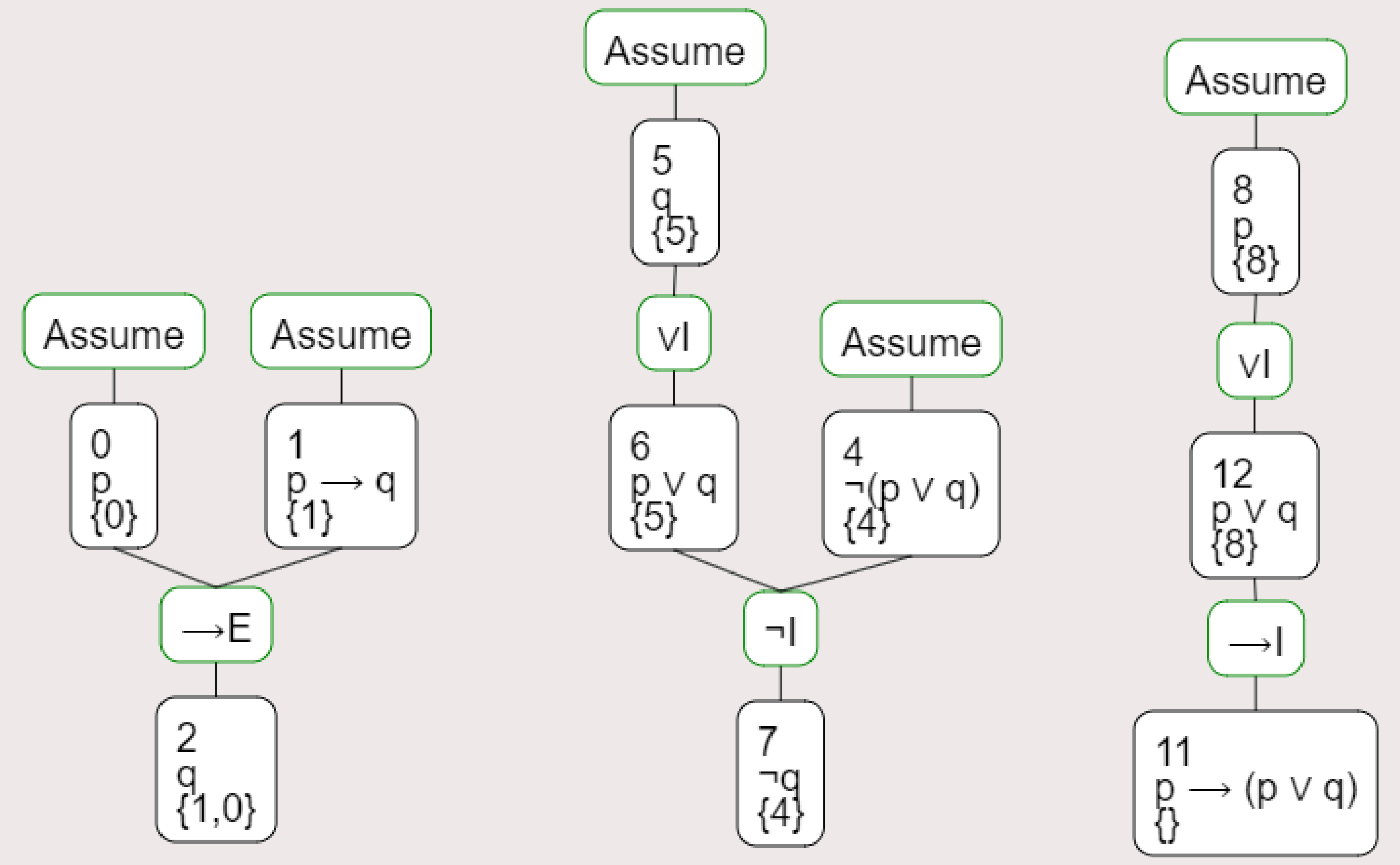


Figure: Three basic natural deduction proof graphs. From left to right Modus Ponens (if elim), $\neg(p \vee q) \vdash \neg q$, and $\vdash p \rightarrow (p \vee q)$

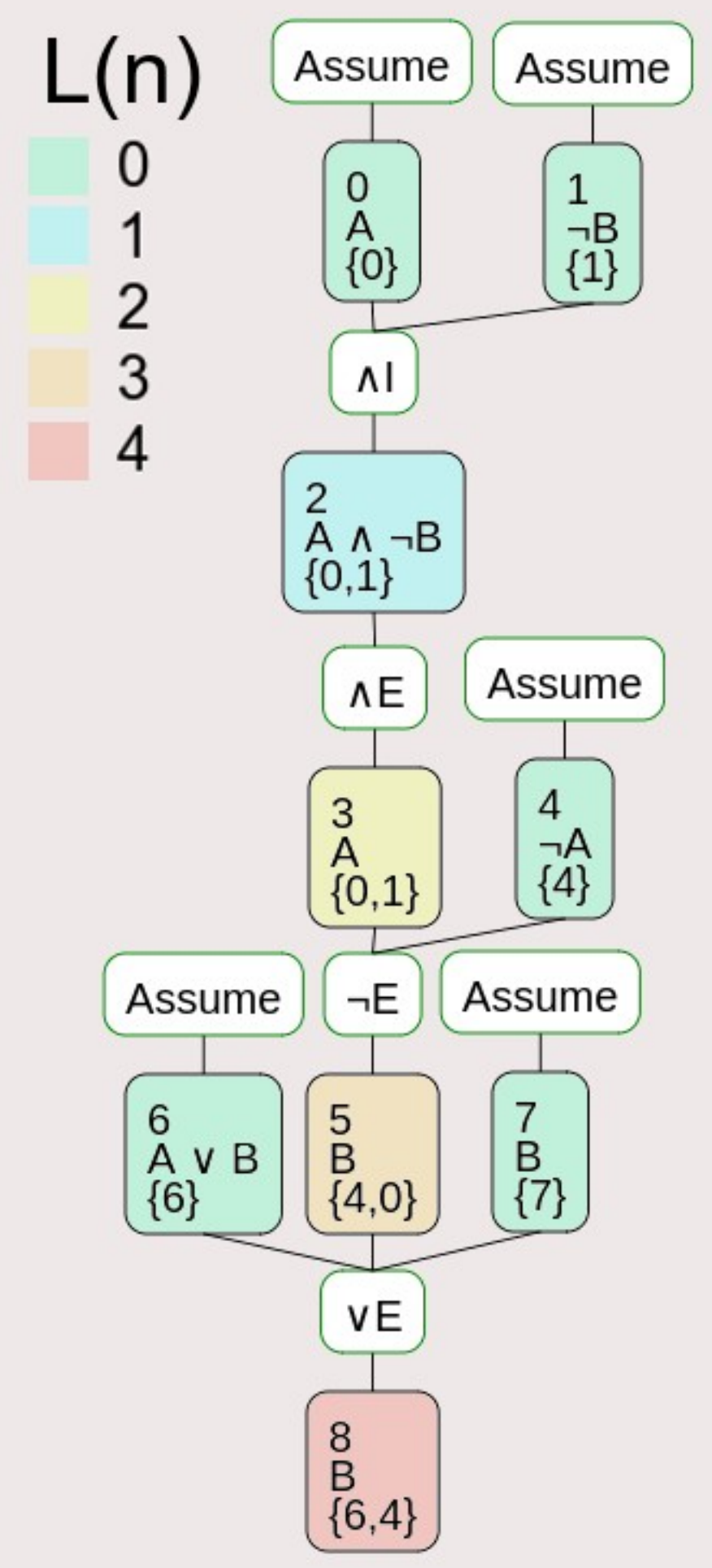
Layering and Serial Solution

We define *layering* for proof graphs, which provides a notion for effectively determining verification dependencies. As seen in Tables 1 and 2, rules depend on sets of assumptions from parent nodes in the previous layers. The layer of a node n in a proof graph is the maximum distance from an assumption to n, formally

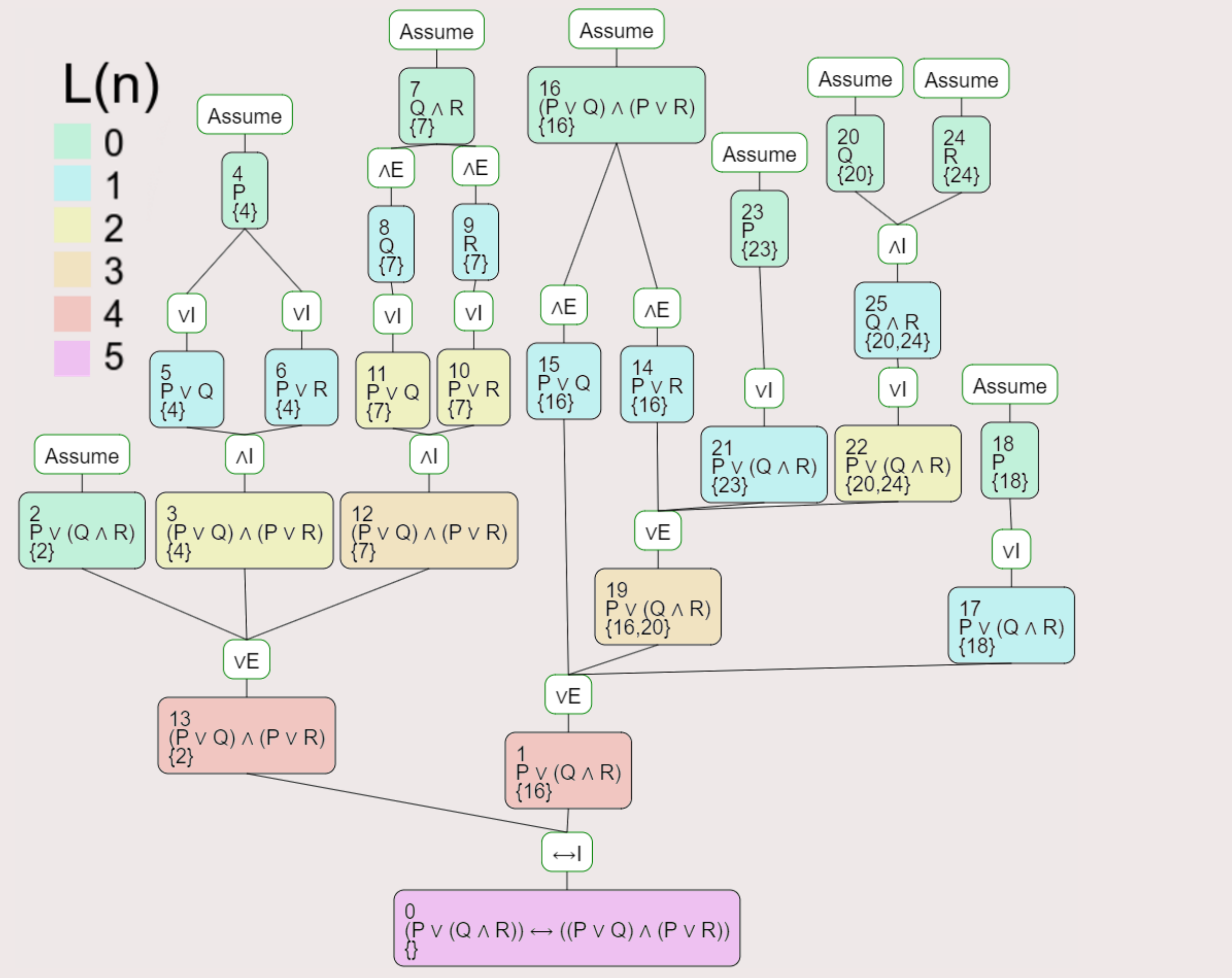
$$L(n) = \begin{cases} 0, & \text{(if assumption)} \\ 1 + \max_{m \in P(n)} L(m), & \text{(otherwise)} \end{cases}$$

Where P is a function mapping a node n to the set of its parents (its incoming edges). For a non-parallel (serial) algorithm, we can begin by computing the layers for all nodes. Once the layers are known, the layers can be iterated over and each node in a layer can be verified according to the rule on its incoming edge.

Figure: A proof that $\neg A, A \vee B \vdash B$ with the layers colored



Parallel Algorithm and Load Balancing



Layer	Nodes
0	2,4,7,16, 23,20,24,18
1	5,6,8,9,15, 14,21,25,17
2	3,11,10, 22
3	12,19
4	1, 13
5	0

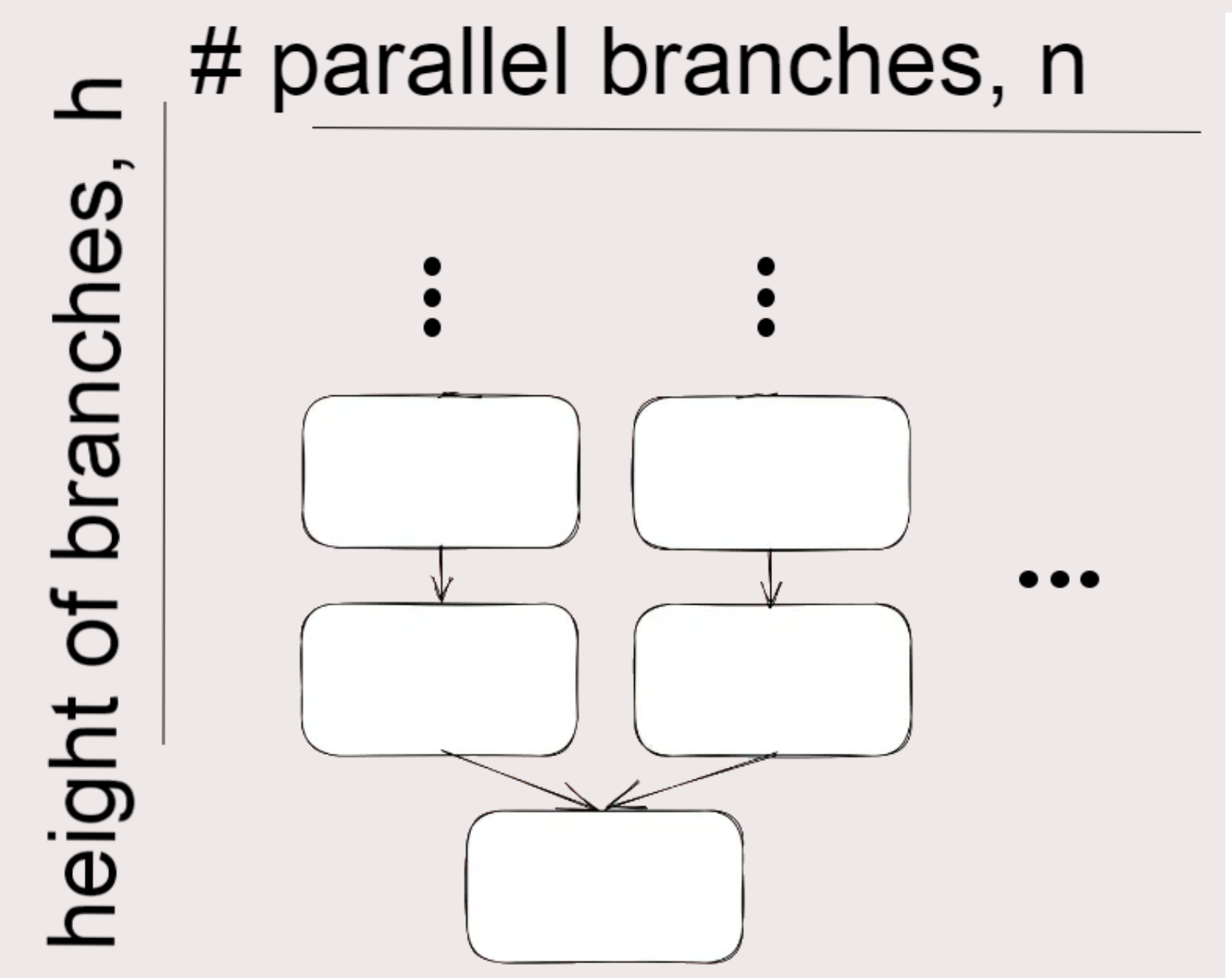
Figure: (Left) A proof of logical or ($\vee$) distributivity over logical and ($\wedge$). (Top) The nodes the left proof grouped by layer.

Non-optimized parallelization: Assuming w parallel ranks (threads), on each layer the nodes are divided equally among the ranks and verified in parallel. For example if $w = 2$, the 8 nodes on layer 0 will be evenly split between the two ranks for verification. Then the 9 nodes on layer 1 will have 4 go to one rank and 5 the other. This process can become inefficient on layers like 3,4,5 under large w as ranks are waiting with nothing to do.

Load Balancing: Syntactic checks are independent of assumption updates on each layer, allowing unused ranks to to skip ahead and syntactically verify the next layer. For example on layer 1 if $w = 2$, the rank that was assigned to verify 4 rather than 5 nodes will additionally perform a syntax check on the first node on layer 2 to balance the workload.

Experiments and Results

We experiment with 5 algorithms: serial verification (Serial), and four variants of parallel verification based on their use of parallel (mpi) or serial (os) layer computation and non-optimized (no) or load balanced (lb) verification algorithm. We showcase only on proof topologies in which parallelization offers benefit, serial will always beat parallel on straight proof topologies.



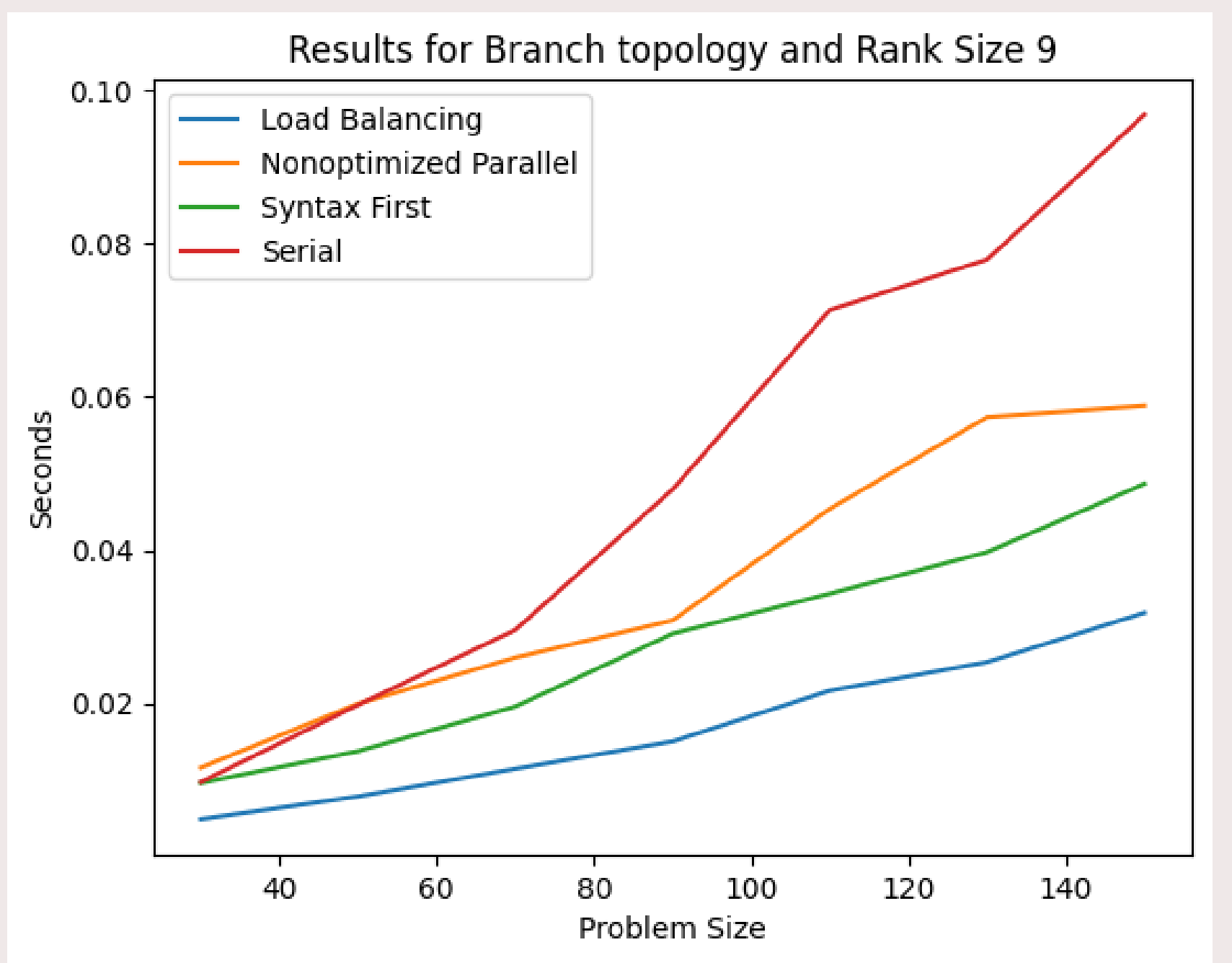
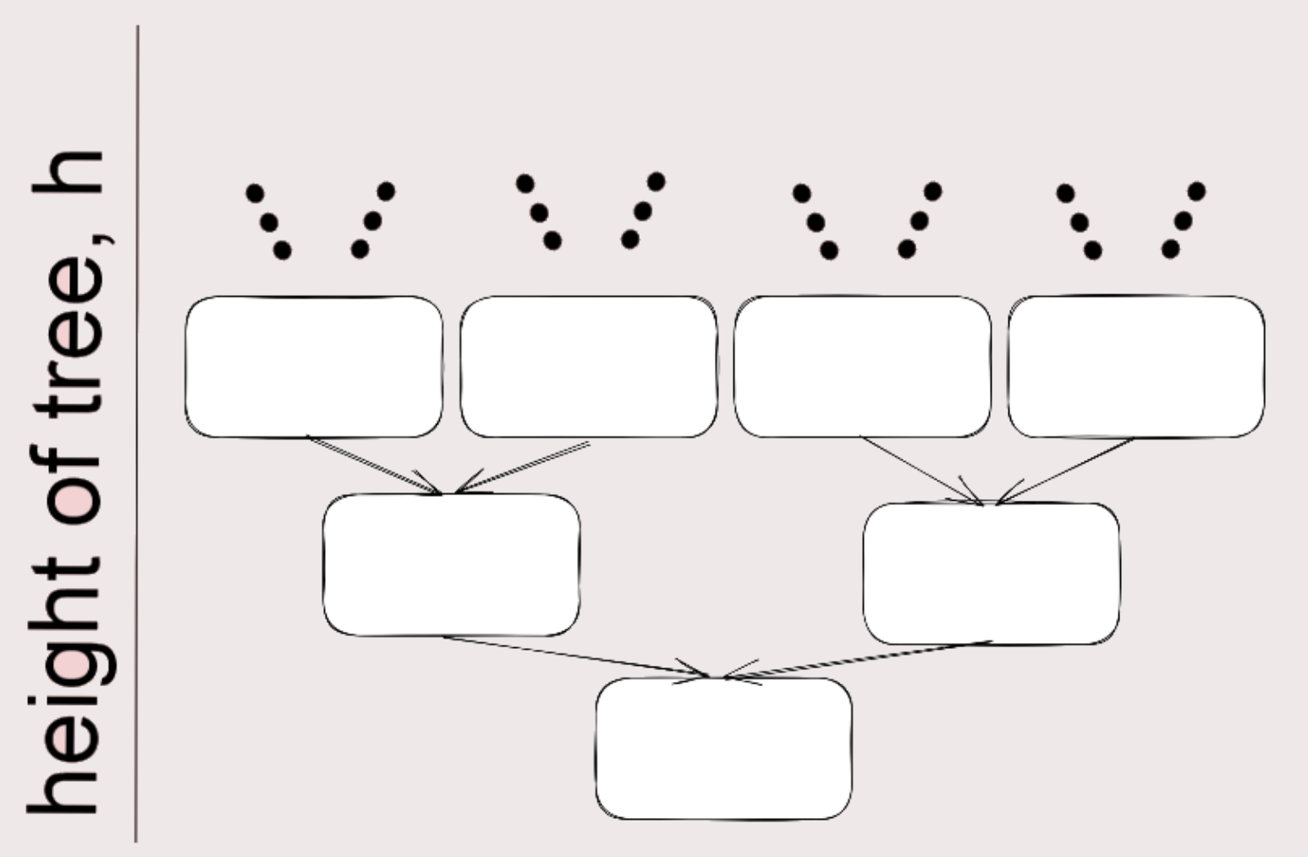


Figure: (Left) The parallel branches proof topology parameterized by the height of each branch h and number of branches n. (Right) Timing results on a parallel branches topology (where branches are "or intro" and joins are "and intro") for $h = 100$ and variable n on the x axis.



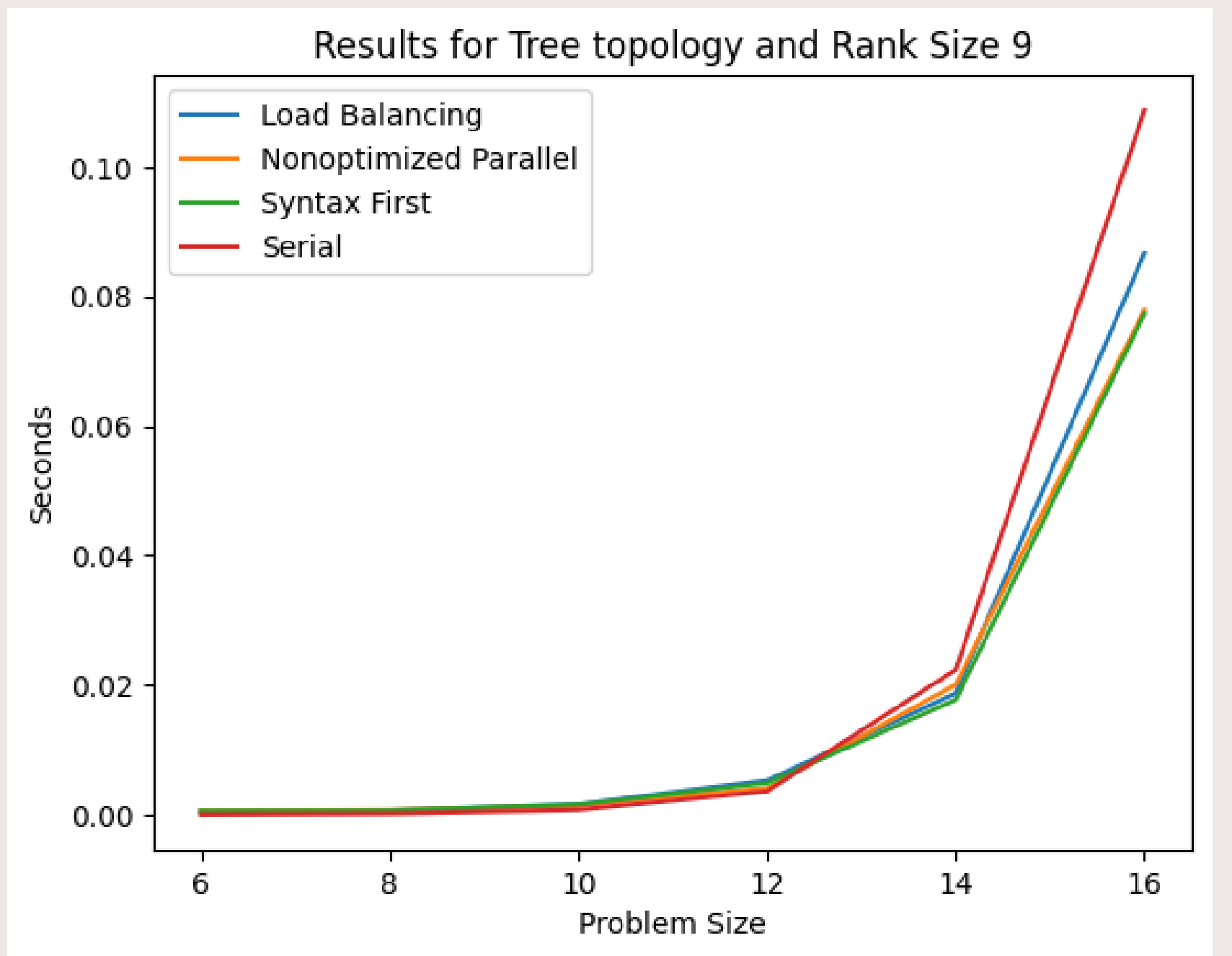


Figure: (Left) The tree proof topology parameterized by the height of the tree h. (Right) Timing results on a tree topology (where joins are "and intro") for a variable tree height, h on the x axis.

On these topologies the parallel algorithms outperforms serial verification as the problem gets larger. Load balancing performs well on the branch topology and that all parallel methods are performant on the tree topology.

Future Work

Other frameworks such as the vampire theorem prover (Riazanov & Voronkov 2002) output proof graphs with more rules, our work could be extended to work with these. Additionally, null models for proof graphs that capture the range of natural deduction proofs would provide a more realistic performance evaluation with respect to our algorithms.

References

Bringsjord, S., Govindarajulu, N. S., Taylor, J. & Bringsjord, A. (2022), *Logic: A Modern Approach*.
Gentzen, G. (1935), 'Untersuchungen über das logische schließen. i', *Mathematische Zeitschrift* pp. 176–210.
Jaśkowski, S. (1934), 'On the rules of suppositions in formal logic'.
Riazanov, A. & Voronkov, A. (2002), 'The design and implementation of vampire', *AI Commun.* p. 91â110.