Strong Bridges for the Circuit Constraint

Implementing and Evaluating Strong Bridge Detection in a Lazy Clause Generation Solver

Author: Martijn van Leest, mjcvanleest@tudelft.nl
Supervisors: Dr. Emir Demirović, Ir. Imko Marijnissen
EEMCS, Delft University of Technology, The Netherlands

1. Introduction

Strongly Connected Component (SCC): Set of nodes that can all reach eachother
Strong bridge: Removing this edge → splits up SCCs
Circuit constraint: Single cycle over all nodes in a graph

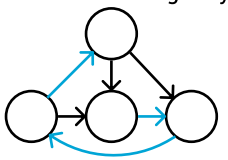


Figure 1: Blue edges are strong bridges, as removing any results in disconnected graph

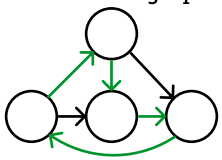


Figure 2: Circuit constraint holds

- Circuit appears in **NP-Hard problems** (e.g. Travelling Salesperson)
- Performance depends propagation algorithm
- Strong propagation is **critical**

Lazy Clause Generation (LCG): Constraint Programming technique combining **constraint propagation** with **clause learning. Explanations** accompany a propagation and answer why a value was removed or fixed.

Quality of explanations is measured by:
- **nogood length** (number of literals)
- **Literal Block Distance** (number of decision levels).

Lower values are better.

2. Objective

Existing methods enforce connectivity or prevent subcycles, but do not identify which edges are required.

Goal: Evaluate the impact of strong bridge detection on the circuit constraint in LCG.

Evaluate based on:
- Search effort (number of conflicts and propagations)
- Explanation Quality (LBD and nogood length)
- Runtime

TU Delft Delft University of Technology

3. Methodology

Key Idea

- Circuit constraint → graph must be **single SCC**
- Removing a strong bridge → **breaks SCC**

→ These edges are **required** in every solution

- **Baseline propagator** prevents subcycles
- **Strong bridge extension**
  - Ensures graph is single SCC
  - Detects strong bridges using reachability checks
  - Enforces these edges
  - Complements baseline

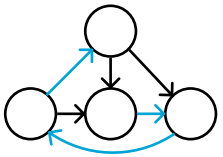


Figure 3: Graph with strong bridges

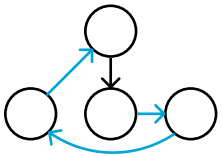
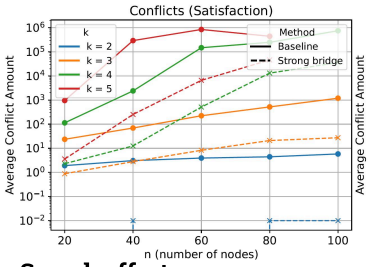


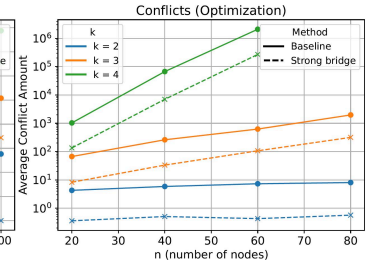
Figure 4: Graph after pruning based on those strong bridges

4. Experimental Setup

- Compare baseline vs strong bridge extension
- Satisfaction (SAT) and optimization (OPT) setting
- Instances vary in size (n) and density (k)
- 100 instances per configuration
- Metrics:
  - Conflicts & propagations
  - Explanation quality (LBD, nogood length)
  - Runtime

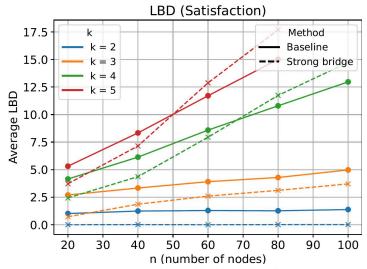
5. Results



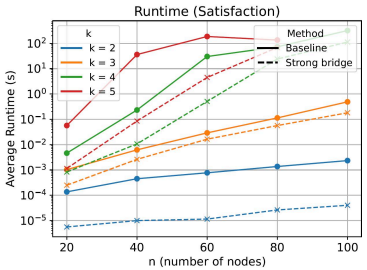


Search effort
- Conflicts reduced by up to 80-99%
- Propagation reduced by 80−99% for dense graphs, ≥40% for sparse graphs

Explanation quality
- Nogoods significantly shorter (up to ~100% in SAT, ~86% in OPT)
- LBD is lower for small instances, but grows with graph size



Runtime
- 40-99% faster in SAT, up to 94% in OPT
- Improvements decrease for larger graphs
- Slowdowns in optimization for k = 3 (-37%)



6. Conclusions

Conclusion:
- Strong bridge detection significantly reduces search effort
- Produces shorter explanations

Trade-off:
- Stronger propagation vs computational overhead
- Larger LBD as problem size grows

Limitations:
- Synthetic instances, fixed search strategy, scalability of results

7. Future Work

Scalability
- Evaluate on more diverse problem types

Effectiveness
- Reduce computational complexity of strong bridge detection
- Construct smaller (more precise) explanations

Understanding
- Use profiling for more in-depth understanding the consequences of strong bridge detection

Exploration
- Combine with other graph connectivity techniques