Enumerating Potential Maximal Cliques via SAT and ASP

Tuukka Korhonen Jeremias Berg Matti Järvisalo

HIIT, Department of Computer Science, University of Helsinki, Finland

IJCAI 2019
Macao
August 16, 2019

Motivation: Potential Maximal Cliques

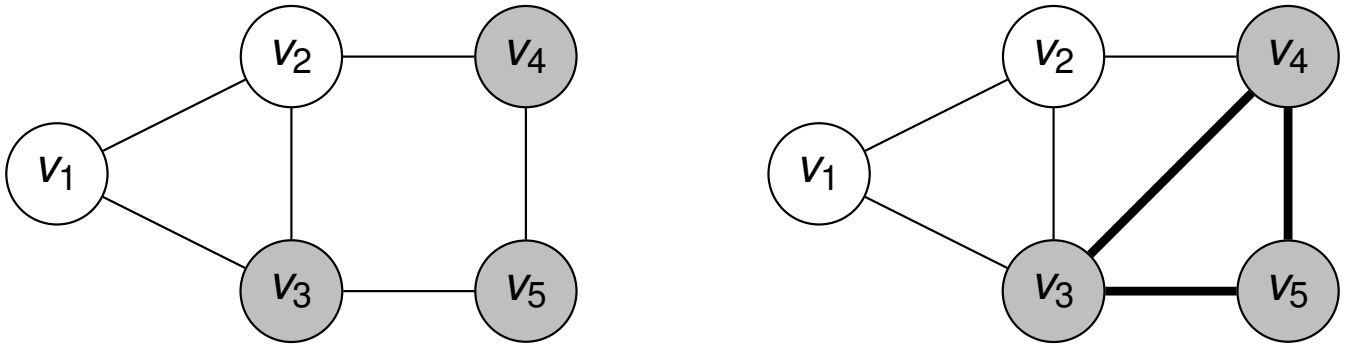
- **Minimal triangulations** define graph decompositions:
  - Tree decomposition
  - Junction tree
  - Phylogenetic tree
  - Hypertree decompositions

Potential maximal cliques (PMCs) are building blocks of minimal triangulations

Motivation: Potential Maximal Cliques

- **Minimal triangulations** define graph decompositions:
  - Tree decomposition
  - Junction tree
  - Phylogenetic tree
  - Hypertree decompositions

  Potential maximal cliques (PMCs) are building blocks of minimal triangulations



Motivation: BT Algorithm

- BT Algorithm Bouchitté and Todinca [2001]
  - Input: Potential maximal cliques
  - Output: Optimal triangulation w.r.t some cost function
  - Runtime: Linear in #PMCs
- Best algorithms in both theory and practice are based on BT Fomin et al. [2008]; Tamaki [2017]; Korhonen et al. [2019].
- Runtime bottleneck: Enumeration of PMCs

Motivation: BT Algorithm

- BT Algorithm Bouchitté and Todinca [2001]
  - Input: Potential maximal cliques
  - Output: Optimal triangulation w.r.t some cost function
  - Runtime: Linear in #PMCs
- Best algorithms in both theory and practice are based on BT Fomin et al. [2008]; Tamaki [2017]; Korhonen et al. [2019].
- Runtime bottleneck: Enumeration of PMCs

Motivation: BT Algorithm

- BT Algorithm Bouchitté and Todinca [2001]
  - Input: Potential maximal cliques
  - Output: Optimal triangulation w.r.t some cost function
  - Runtime: Linear in #PMCs
- Best algorithms in both theory and practice are based on BT Fomin et al. [2008]; Tamaki [2017]; Korhonen et al. [2019].
- Runtime bottleneck: Enumeration of PMCs

Contributions

New ideas

- Declarative encodings for enumeration of PMCs
- Three variants: direct SAT, lazy SAT and ASP

Advantages

- Enumerate only relevant PMCs
  - $|PMC| \leq k$ if treewidth $< k$
- Solvers can avoid the worst case time complexity in practice

Experimental evaluation

- Applications: Computation of treewidth (TW) and generalized hypertreewidth (GHTW)
- Improvement over the original PMC enumeration algorithm

Contributions

New ideas

- Declarative encodings for enumeration of PMCs
- Three variants: direct SAT, lazy SAT and ASP

Advantages

- Enumerate only relevant PMCs
  - $|PMC| \leq k$ if treewidth $< k$
- Solvers can avoid the worst case time complexity in practice

Experimental evaluation

- Applications: Computation of treewidth (TW) and generalized hypertreewidth (GHTW)
- Improvement over the original PMC enumeration algorithm

Contributions

New ideas

- Declarative encodings for enumeration of PMCs
- Three variants: direct SAT, lazy SAT and ASP

Advantages

- Enumerate only relevant PMCs
  - $|PMC| \leq k$ if treewidth $< k$
- Solvers can avoid the worst case time complexity in practice

Experimental evaluation

- Applications: Computation of treewidth (TW) and generalized hypertreewidth (GHTW)
- Improvement over the original PMC enumeration algorithm

Encodings

Definition of PMC

Implicit definition

- Set of vertices that is maximal clique in some minimal triangulation

Connectedness outside of K

- $\exists$Path between u and v with no *intermediate* vertex in K

Explicit definition

- A set of vertices $K \subset V$ is a PMC if:
  1. Any two vertices $u, v \in K$ are connected outside of K
  2. No vertex in $V \setminus K$ is connected to all vertices of K outside of K

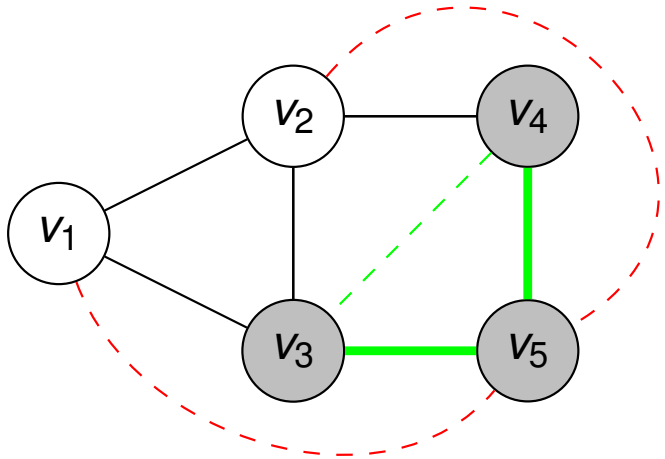
Definition of PMC

Implicit definition

- Set of vertices that is maximal clique in some minimal triangulation

Connectedness outside of K

- $\exists$Path between u and v with no *intermediate* vertex in K



Explicit definition

- A set of vertices $K \subset V$ is a PMC if:

1. Any two vertices $u, v \in K$ are connected outside of K
2. No vertex in $V \setminus K$ is connected to all vertices of K outside of K

Definition of PMC: Constraint encoding

- Binary variables: P_i and C_{ij}
  - $P_i = 1$ iff v_i is included in the PMC
  - $C_{ij} = 1$ iff v_i and v_j are connected outside of the PMC
- Constraints: Two conditions for PMC
  1. $(P_i \wedge P_j) \rightarrow C_{ij}$ (pairwise connected)
  2. $\neg P_i \rightarrow \bigvee_{j=1}^{n} (P_j \wedge \neg C_{ij})$ (no vertex outside connected to all)
- Hard part: Encoding connectivity

Definition of PMC: Constraint encoding

- Binary variables: P_i and C_{ij}
  - $P_i = 1$ iff v_i is included in the PMC
  - $C_{ij} = 1$ iff v_i and v_j are connected outside of the PMC
- Constraints: Two conditions for PMC
  1. $(P_i \wedge P_j) \rightarrow C_{ij}$ (pairwise connected)
  2. $\neg P_i \rightarrow \bigvee_{j=1}^{n}(P_j \wedge \neg C_{ij})$ (no vertex outside connected to all)
- Hard part: Encoding connectivity

Definition of PMC: Constraint encoding

- Binary variables: P_i and C_{ij}
  - $P_i = 1$ iff v_i is included in the PMC
  - $C_{ij} = 1$ iff v_i and v_j are connected outside of the PMC
- Constraints: Two conditions for PMC
  1. $(P_i \wedge P_j) \rightarrow C_{ij}$ (pairwise connected)
  2. $\neg P_i \rightarrow \bigvee_{j=1}^{n}(P_j \wedge \neg C_{ij})$ (no vertex outside connected to all)
- Hard part: Encoding connectivity

Encoding connectivity

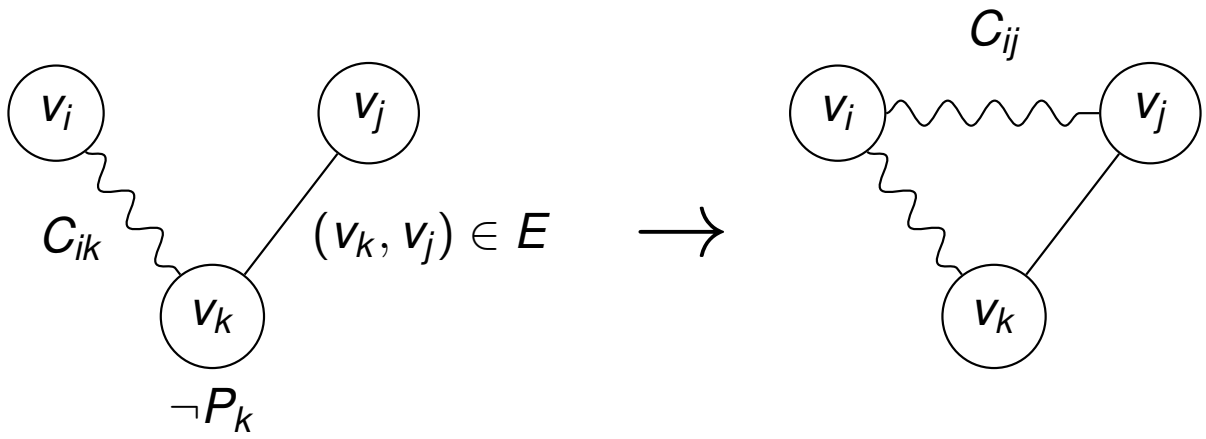
- $C_{ij} = 1$ if and only if v_i and v_j are connected outside the PMC
- If-direction is easy without additional variables

$$(C_{ik} \wedge (v_k, v_j) \in E \wedge \neg P_k) \rightarrow C_{ij}$$

Encoding connectivity

- $C_{ij} = 1$ if and only if v_i and v_j are connected outside the PMC
- If-direction is easy without additional variables

$$(C_{ik} \wedge (v_k, v_j) \in E \wedge \neg P_k) \rightarrow C_{ij}$$



Encoding connectivity: Only if

Three approaches for ”only if”-direction

1. Path-length encoding in SAT
   - Existence of a shorter path required for a longer path
   - $O(n^3)$ additional variables
2. ASP semantics
   - Existence of a path must have external support
   - No additional variables or constraints
3. Lazy SAT

Encoding connectivity: Only if

Three approaches for ”only if”-direction

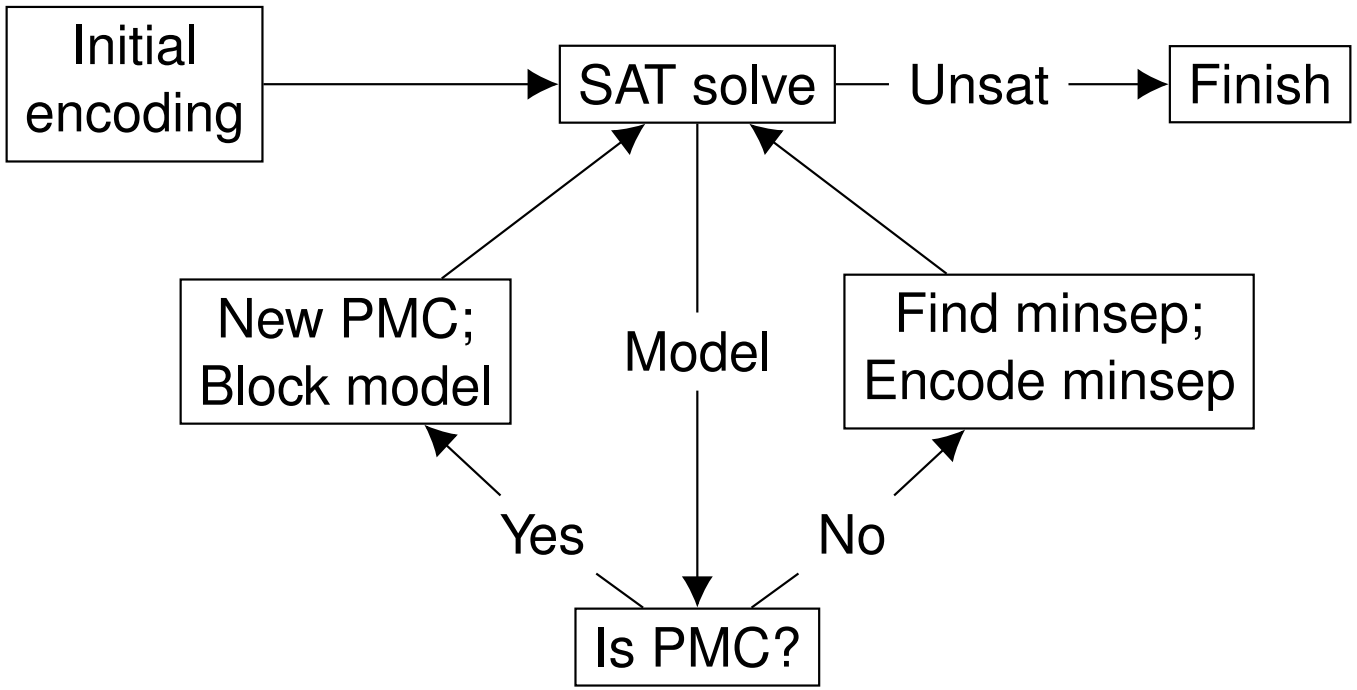
1. Path-length encoding in SAT
   - Existence of a shorter path required for a longer path
   - $O(n^3)$ additional variables
2. ASP semantics
   - Existence of a path must have external support
   - No additional variables or constraints
3. Lazy SAT

Encoding connectivity: Only if

Three approaches for ”only if”-direction

1. Path-length encoding in SAT
   - Existence of a shorter path required for a longer path
   - $O(n^3)$ additional variables
2. ASP semantics
   - Existence of a path must have external support
   - No additional variables or constraints
3. Lazy SAT

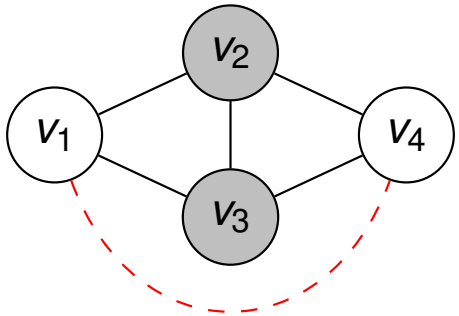
Lazy SAT



- Initialize SAT formula without ”only if”-encoding
- For each model, check if it represents a PMC

Lazy SAT

- If the found model does not represent a PMC:
  - $C_{ij} = 1$ for some pair v_i, v_j even though they are not connected
  - A *minimal separator* inside the PMC proves they are not connected

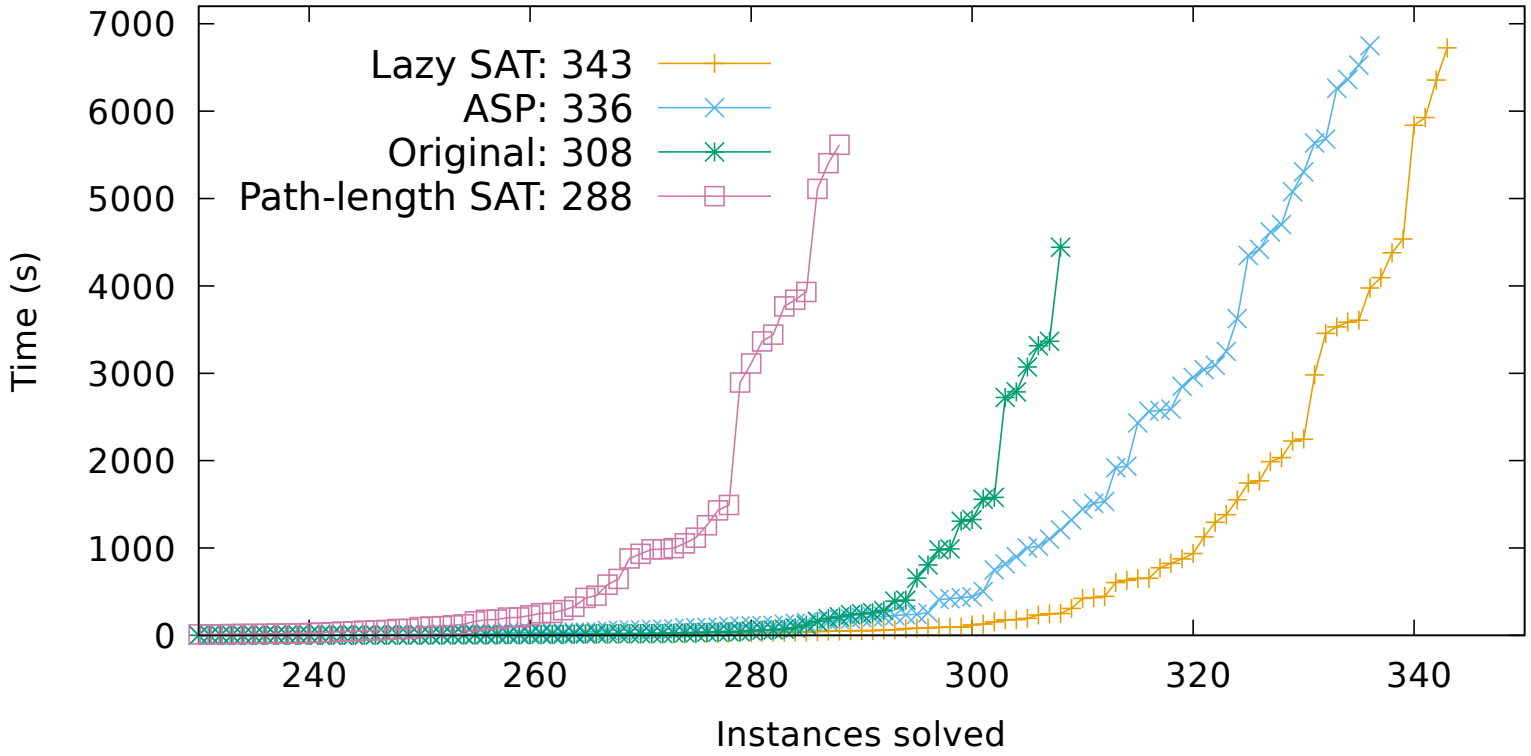


Encoding

- Minimal separator is active if it is contained in the PMC
  - $\left(\bigwedge_{v_i \in S} P_i\right) \rightarrow M_S$
- An active minimal separator implies non-connectivity
  - $\bigwedge_{i,j:\ S \text{ separates } v_i, v_j} M_S \rightarrow \neg C_{ij}$

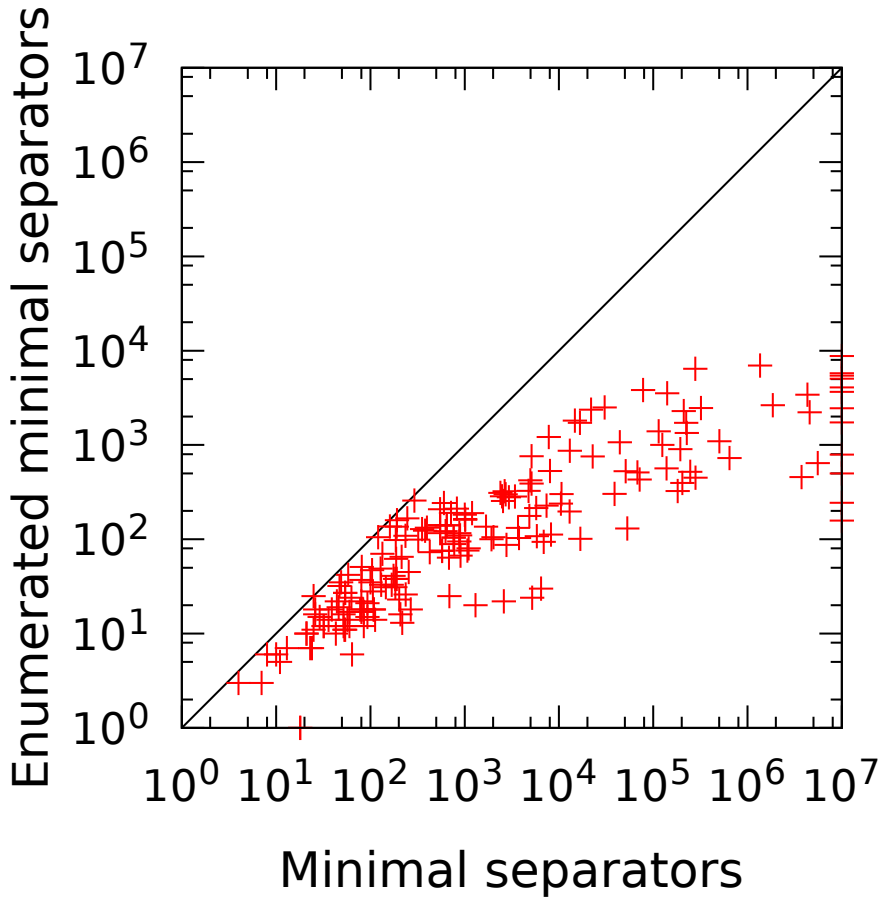
Experiments

Experiments



- Lazy SAT and ASP improve BT implementation on treewidth computation

Experiments: Minimal separators



- Lazy SAT is efficient even if the number of total minimal separators is large
- The original PMC enumeration algorithm requires enumeration of all minimal separators

Summary

- Enumerating potential maximal cliques is important for computing graph decompositions
- Proposed encodings for enumerating PMCs with SAT and ASP solvers
- The hard part of encoding is the only if direction of connectivity constraints
  - Three different variants
- Lazy SAT and ASP outperform the original PMC enumeration algorithm

The end

Thank you for your attention!

Bibliography

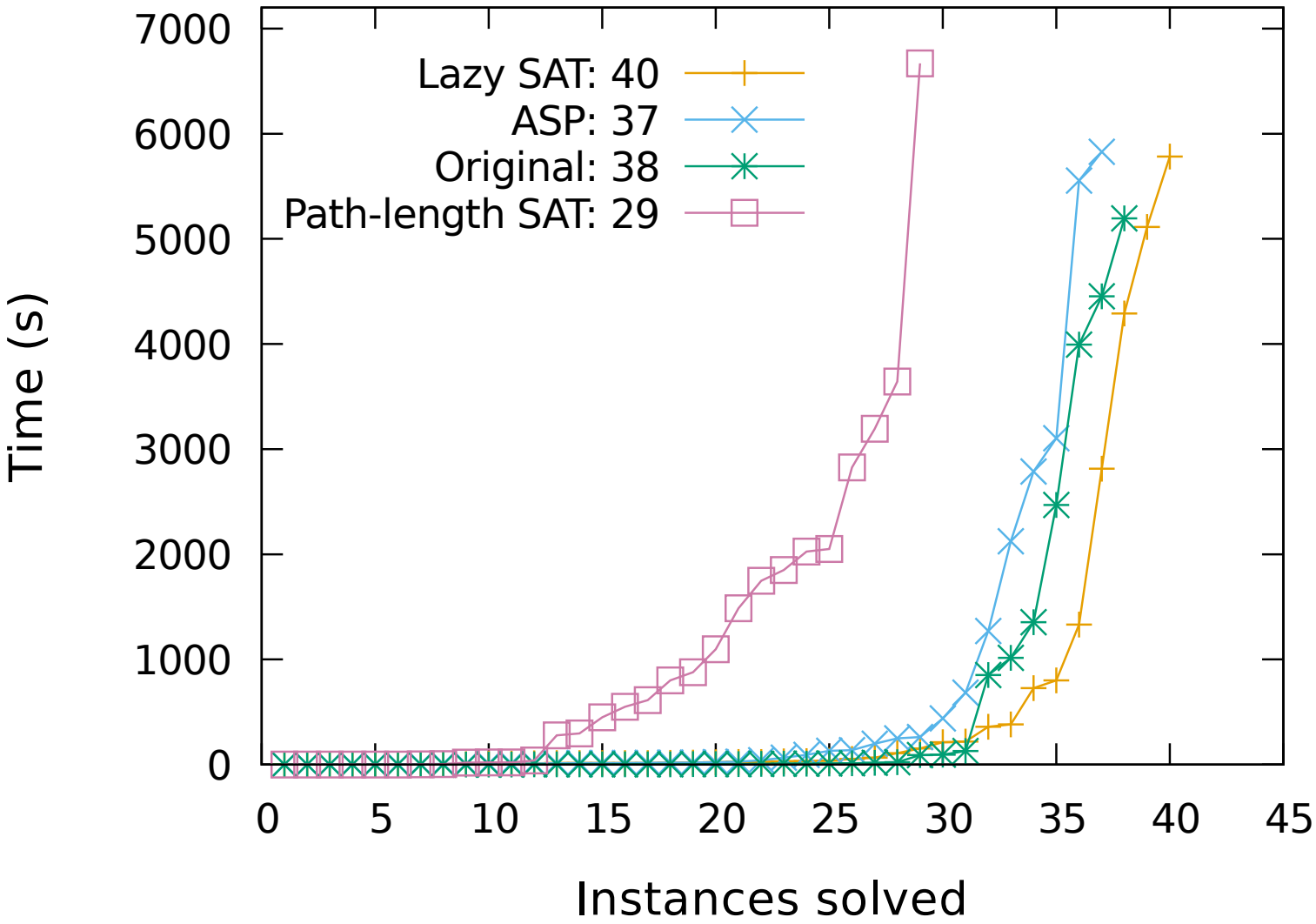
Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1): 212–232, 2001.

Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38(3):1058–1079, 2008.

Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Solving graph problems via potential maximal cliques: An experimental evaluation of the Bouchitté-Todinca algorithm. *ACM J. Exp. Alg.*, 24(1:9), 2019.

Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In *ESA*, volume 87 of *LIPIcs*, pages 68:1–68:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

GHTW results



Melon graphs

Table: Runtime on melon graphs with $3n + 2$ vertices.

n	Original (s)	Lazy SAT (s)
6	0.46	0.04
7	3.27	0.06
8	26.80	0.06
9	261.20	0.09
10	2576.73	0.09
20	TO	0.38
50	TO	3.52
100	TO	39.00
200	TO	384.95

Complexity

- BT algorithm - $O(\#PMC \cdot n^3)$
- Original PMC enumeration - $O(n^2 \cdot m \cdot \#minsep^2)$
- Path-length SAT - $O(n^3)$ variables, $O(n^2 m)$ clauses
- Lazy SAT - $O(n^2 + \#minsep)$ variables, $O(nm + \#minsep \cdot n^2)$ clauses

Instance sources

- Treewidth 589 instances:
  - PACE 2016, 2017; TW and min-fill
  - DIMACS
  - bnlearn
- GHTW 265 instances:
  - `https://www.dbai.tuwien.ac.at/proj/hypertree/downloads.html`

Comparison to other solvers

- Treewidth:
  - Lazy SAT: 343
  - ASP: 336
  - Triangulator (original): 308
  - PIDDT (Tamaki): 467
  - PIDDT (Bannach et at.): 457
- GHTW:
  - Lazy SAT: 40
  - ASP: 37
  - Triangulator (original): 38
  - FraSMT: 25