Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)

Enumerating Potential Maximal Cliques via SAT and ASP

Tuukka Korhonen , **Jeremias Berg** and **Matti Järvisalo**
HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract

The Bouchitté-Todinca algorithm (BT), operating dynamic programming over the so-called potential maximal cliques (PMCs), yields a practically efficient approach to treewidth and generalized hypertreewidth. The enumeration of PMCs is a scalability bottleneck for BT in practice. We propose the use of declarative solvers for PMC enumeration as a substitute for the specialized PMC enumeration algorithms employed in current BT implementations. The presented Boolean satisfiability (SAT) and answer set programming (ASP) based PMC enumeration approaches open up new possibilities for improving the efficiency of BT in practice.

1 Introduction

Bounded treewidth of graphs and its generalizations to hypergraphs are central structural restrictions that characterize tractable fragments in a variety of settings, from constraint satisfaction [Freuder, 1985; Dechter, 2006], discrete optimization [Bodlaender and Koster, 2008] and knowledge representation [Gottlob *et al.*, 2010] to probabilistic reasoning [Lauritzen and Spiegelhalter, 1988; Darwiche, 2003]. Determining treewidth and generalized hypertreewidth is NP-hard [Arnborg *et al.*, 1987; Gottlob *et al.*, 2009], which makes the development of practical algorithms for computing the associated optimal decompositions a non-trivial challenge. In answer to this challenge, various types of exact algorithms, ranging from specialized algorithms [Gogate and Dechter, 2004; Moll *et al.*, 2012; Tamaki, 2017; Korhonen *et al.*, 2019] to declarative approaches [Samer and Veith, 2009; Berg and Järvisalo, 2014; Lodha *et al.*, 2016; Lodha *et al.*, 2017; Fichte *et al.*, 2018; Ganian *et al.*, 2019], have been recently proposed for treewidth [Samer and Veith, 2009; Berg and Järvisalo, 2014; Tamaki, 2017; Korhonen *et al.*, 2019], its generalizations to hypergraphs [Moll *et al.*, 2012; Fichte *et al.*, 2018; Korhonen *et al.*, 2019], and related width notions [Lodha *et al.*, 2016; Lodha *et al.*, 2017; Ganian *et al.*, 2019]. Among these approaches is the BT algorithm of Bouchitté and Todinca (2001) which operates dynamic programming over the so-called potential maximal cliques (PMCs). Originally proposed for treewidth, BT has recently been shown to be a practically efficient approach to treewidth as well as to various other NP-hard optimization problems related to graph triangulations [Tamaki, 2017; Dell *et al.*, 2018; Korhonen *et al.*, 2019], outperforming declarative approaches to determining treewidth, and constituting currently the most effective exact approach to determining generalized hypertreewidth [Korhonen *et al.*, 2019].

The enumeration of PMCs, using a specialized enumeration algorithm [Bouchitté and Todinca, 2002], has been acknowledged as a bottleneck both in terms of worst-case analysis [Fomin *et al.*, 2008] and scalability of BT in practice [Korhonen *et al.*, 2019]. The question of whether PMCs can be enumerated with polynomial delay is currently open [Bodlaender *et al.*, 2006]. Thus developing more effective approaches to enumerating PMCs is both a non-trivial challenge and a key to improving the scalability of BT in practice.

We propose the use declarative solvers for PMC enumeration as a substitute for the specialized PMC enumeration algorithms employed in current BT implementations. We develop both direct Boolean satisfiability (SAT) [Biere *et al.*, 2009] and answer set programming (ASP) [Gelfond and Lifschitz, 1988; Niemelä, 1999] encodings as well as a SAT-based lazy approach for the task of PMC enumeration. We integrate these constraint-based enumeration approaches effectively to an iterative variant of BT which at the same time allows for avoiding the enumeration on all PMCs on graphs with low (tree or generalized hypertree)width. As a result, we obtain a novel iterative hybrid variant of BT, combining SAT solving and dynamic programming. Empirically, we show that integrating the proposed approaches into a recent BT implementation compares favorably with the use of specialized enumeration on the tasks of determining treewidth and generalized hypertreewidth. Furthermore, harnessing declarative solvers for enumerating PMCs opens up further opportunities for improving the efficiency of BT approaches to triangulation-based graph optimization problems through further improvements in SAT and ASP solvers.

2 Preliminaries

We consider undirected, simple graphs. We denote by $V(G)$ and $E(G)$ the set of vertices and edges, resp., of a given graph G. The neighbourhood $\mathrm{N}(v)$ of a vertex $v \in V(G)$ consists of vertices u with $\{u, v\} \in E(G)$. For a set $S \subset V(G)$, $\mathrm{N}(S) = \cup_{v \in S} \mathrm{N}(v) \setminus S$. A graph G is chordal if every cycle of length at least 4 has a chord, i.e., an edge joining two non-

adjacent vertices in the cycle. A *triangulation* H of G is a chordal graph that contains G, specifically, a graph for which $V(H) = V(G)$ and $E(G) \subset E(H)$. A triangulation H of G is minimal if no proper subgraph of H is a triangulation of G. We denote the set of minimal triangulations of G by $\mathrm{MT}(G)$.

For a graph G, the graph $G[S]$ induced by $S \subset V(G)$ has $V(G[S]) = S$ and $E(G[S]) = E(G) \cap \{\{u, v\} \mid u \in S, v \in S\}$. We use the notation $G \setminus S = G[V(G) \setminus S]$. A set $\omega \subset V(G)$ is a clique (of G) if $G[\omega]$ is complete, and a maximal clique if no other clique ω' satisfies $\omega \subsetneq \omega'$. We denote the set of maximal cliques of G by $\mathrm{MC}(G)$.

The width $\mathrm{W}(H)$ of a triangulation $H \in \mathrm{MT}(G)$ of G is $\max_{\omega \in \mathrm{MC}(H)}\{|\omega| - 1\}$, i.e., the size of the largest clique of H minus one. The **treewidth** $\mathrm{TW}(G)$ of a graph G is $\min_{H \in \mathrm{MT}(G)}\{\mathrm{W}(H)\}$, i.e., the minimum width over all triangulations of G.

A set $\Omega \subset V(G)$ is a **potential maximal clique** (PMC) of G if there is a $H \in \mathrm{MT}(G)$ such that $\Omega \in \mathrm{MC}(H)$. We denote the set of potential maximal cliques of G by $\Pi(G)$, and the set of PMCs of size at most k by $\Pi_k(G)$. For a subset $\Pi_s \subset \Pi(G)$, we denote by $\mathrm{MT}(G, \Pi_s)$ the minimal triangulations H of G for which $\mathrm{MC}(H) \subset \Pi_s$; and by $\mathrm{TW}(G, \Pi_s)$ the minimum width of all triangulations in $\mathrm{MT}(G, \Pi_s)$; $\mathrm{TW}(G, \Pi_s) = \infty$ if $\mathrm{MT}(G, \Pi_s) = \emptyset$. Note that $\mathrm{MT}(G, \Pi(G)) = \mathrm{MT}(G)$, $\mathrm{TW}(G, \Pi(G)) = \mathrm{TW}(G)$, and $\mathrm{TW}(G) = \mathrm{TW}(G, \Pi_k(G)) \leq k-1$ if $\mathrm{TW}(G, \Pi_k(G)) < \infty$.

Example 1 *Consider the graph G in Figure 1 left: G is not chordal, witnessed by the cycle (s, u, b, t, v). One of its minimal triangulations $H \in \mathrm{MT}(G)$ is shown in Figure 1 right. The sets $\{a, s, v\}$ and $\{s, v, u\}$ are examples of maximal cliques of H and thus also examples of PMCs of G. As the largest clique of H has 3 vertices, we have $H \in \mathrm{MT}(G, \Pi_3(G))$ and $\mathrm{W}(H) = 2$.*

We denote the set of connected components of G by $\mathcal{C}(G)$. For a $S \subset V(G)$, a component $C \in \mathcal{C}(G \setminus S)$ is *full* wrt S if $\mathrm{N}(C) = S$, and S is a minimal separator of G if $G \setminus S$ has at least two full components wrt S. We denote the set of minimal separators of G by $\Delta(G)$. Alternatively, a set S is an $\{u, v\}$-separator (or separates the vertices u and v) if u and v are not connected in $G \setminus S$ and is further minimal if no $S' \subsetneq S$ separates u and v. The set $\Delta(G)$ of minimal separators contains all sets S that are minimal separators of some pair of vertices u and v. Note that several different vertex-pairs can share a minimal separator, and that minimal separators of different vertex-pairs can be subsets of each other.

We will use the following characterization of PMCs.

Proposition 1 ([Bouchitté and Todinca, 2001]) *A set $K \subset V(G)$ is a PMC of a graph G if and only if (i) $G \setminus K$ has no full components associated to K, and (ii) completing $\mathrm{N}(C)$ into a clique for each $C \in \mathcal{C}(G \setminus K)$ makes K a clique.*

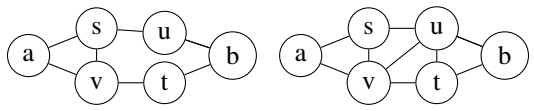

Figure 1: An example graph (left) and one of its triangulation (right)

In particular, the conditions (i) and (ii) of Proposition 1 can be stated in terms of connectivity outside of specific sets of vertices. Given a set $K \subset V(G)$ and $t, u \in V(G)$, the vertices t and u are connected outside K if there is a path $(t, v_1, \ldots, v_n, u)$ in G such that $v_i \in V(G) \setminus K$ for all $i = 1, \ldots, n$. Note that t and u can be connected outside of K even if they are in K. Thus Proposition 1 can be restated as follows for a basis of declarative encodings of PMCs.

Corollary 1 *A set $K \subset V(G)$ is a PMC of a graph G if and only if (i) for every vertex $v \in V(G) \setminus K$, there is a vertex $u \in K$ such that v and u are not connected outside K and (ii) any two vertices $v, u \in K$ are connected outside of K.*

Example 2 *Consider the graph G in Figure 1 (left) and let $K = \{s, v, u\}$. The graph $G \setminus K$ has 2 connected components: $\mathcal{C}(G \setminus K) = \{\{a\}, \{t, b\}\}$. Neither component is full wrt K, since $\mathrm{N}(\{t, b\}) = \{v, u\} \neq K \neq \{s, v\} = \mathrm{N}(\{a\})$. Thus K is not a minimal separator. One minimal $\{a, b\}$-separator within K is $\{s, v\}$. Completing $\mathrm{N}(\{t, b\})$ to a clique adds the edge $\{v, u\}$, making K a clique. Thus K satisfies the conditions of Proposition 1 and hence $K \in \Pi(G)$. In terms of Corollary 1, a is not connected to u outside of K, and neither of t and b is connected to s outside of K. On the other hand, vertices in K are connected pairwise outside of K. Finally, no set containing only 2 vertices satisfies condition (i) of Corollary 1. Hence $\Pi_2(G) = \emptyset$ and $\mathrm{MT}(G, \Pi_2(G)) = \emptyset$, and thus $\mathrm{TW}(G, \Pi_2(G)) = \infty$ and $\mathrm{TW}(G) = \mathrm{TW}(G, \Pi_3(G)) = 2$.*

Hypergraphs generalize graphs by allowing arbitrary subsets of vertices as (hyper)edges. We denote the set of vertices and edges of a hypergraph $\mathcal{G}$ by $V(\mathcal{G})$ and $E(\mathcal{G})$, resp. The primal graph $\textsc{Prim}(\mathcal{G})$ of $\mathcal{G}$ has $V(\textsc{Prim}(\mathcal{G})) = V(\mathcal{G})$ and $E(\textsc{Prim}(\mathcal{G})) = \{\{u, v\} \mid \exists e \in E(\mathcal{G}), \{u, v\} \subset e\}$. For a subset $K \subset V(\mathcal{G})$, a set $E \subset E(\mathcal{G})$ is an edge cover of K if $K \subset \cup E$. We denote the size of the smallest edge cover of K by $\mathrm{COV}_{\mathcal{G}}(K)$. For a hypergraph $\mathcal{G}$, the width of a triangulation $H \in \mathrm{MT}(\textsc{Prim}(\mathcal{G}))$ is $\mathrm{W}(H) = \max_{\omega \in \mathrm{MC}(H)}\{\mathrm{COV}_{\mathcal{G}}(\omega)\}$, and can be determined by computing the smallest edge covers of each clique $\omega \in \mathrm{MC}(H)$. The **generalized hypertreewidth** of $\mathcal{G}$ is $\mathrm{GHTW}(\mathcal{G}) = \min_{H \in \mathrm{MT}(\textsc{Prim}(\mathcal{G}))}\{\mathrm{W}(H)\}$ [Moll *et al.*, 2012; Gottlob *et al.*, 2002; Grohe and Marx, 2014]. Note that, as all edges of $\textsc{Prim}(\mathcal{G})$ are over two vertices, definitions related to (non-hyper) graphs are applicable via the primal graph. In particular, $\mathrm{MT}(\textsc{Prim}(\mathcal{G}), \Pi_s)$ contains all of the triangulations H of $\textsc{Prim}(\mathcal{G})$ for which $\mathrm{MC}(H) \subset \Pi_s$ and $\mathrm{GHTW}(\mathcal{G}, \Pi_s) = \min_{H \in \mathrm{MT}(\textsc{Prim}(\mathcal{G}), \Pi_s)}\{\mathrm{W}(H)\}$.

3 The Bouchitté-Todinca Algorithm

Enumeration of potential maximal cliques forms the basis of the BT algorithm [Bouchitté and Todinca, 2001] which gives the best known exact algorithm for treewidth in terms of worst-case analysis [Fomin and Villanger, 2008] and yields a competitive approach to various triangulation-related graph optimization problems in practice [Tamaki, 2017; Dell *et al.*, 2018; Korhonen *et al.*, 2019]. In the following we overview BT focusing on treewidth; we refer to [Bouchitté and Todinca, 2001; Fomin *et al.*, 2008] for further details. We will

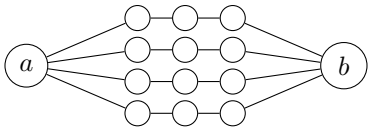

Figure 2: MELON$_4$

explain later in Section 5 how BT and the constraint-based enumeration approaches proposed in this work are adapted for generalized hypertreewidth.

The BT algorithm works in two phases: (i) $\Pi(G)$ is computed, i.e., the potential maximal cliques of G are enumerated (ENUM-PMC); (ii) dynamic programming over $\Pi(G)$ is used to determine the treewidth of G (BT-DP). Phase (ii) decomposes the computation of $\mathrm{TW}(G, \Pi(G))$ into the computation of $\mathrm{TW}(G[S \cup C], \Pi(G))$ for minimal separators $S \in \Delta(G)$ and full components $C \in \mathcal{C}(G \setminus S)$ [Bouchitté and Todinca, 2001; Fomin *et al.*, 2008]. The enumeration of PMCs (the first phase) in the original BT algorithm works by inductively defining $\Pi(G)$ based on $\Pi(G \setminus \{v\})$ for some $v \in V(G)$ and the minimal separators $\Delta(G)$ of the graph [Bouchitté and Todinca, 2002]. This requires enumeration of $\Delta(G)$, using the algorithm of Berry *et al.* (1999).

The dynamic programming phase of BT runs in time $O(poly(n) \cdot |\Pi(G)|)$, where n is the number of vertices. There are no known algorithms for enumerating $\Pi(G)$ with similar time complexity [Bodlaender *et al.*, 2006]. The enumeration phase of the original BT algorithm has time complexity $\Omega(poly(n) \cdot |\Delta(G)|^2)$. This suggests that the bottleneck of the overall efficiency of the BT algorithm is the enumeration of PMCs, which is also the case in practice [Korhonen *et al.*, 2019]. This motivates developing alternative PMC enumeration approaches compatible with BT.

In terms of lower bounds for PMCs enumeration, a concrete example of a graph class with an exponential number of minimal separators is the "melon" graphs $\{\text{MELON}_n\}$, which consist of n paths of length three between two distinct vertices a and b; see Fig. 2 for MELON$_4$. In particular, it has been shown that $|\Delta(\text{MELON}_n)| = \Omega(3^n)$ [Fomin *et al.*, 2008]. Intuitively, the exponentiality is due to the fact that each way of selecting a single vertex from each of the paths forms a minimal separator. Thus any algorithm that enumerates all minimal separators, including the original BT algorithm, will have an unpractical running time for even small melon graphs such as MELON$_{50}$. The declarative approaches to PMC enumeration proposed in this work solve MELON$_{50}$ in a few seconds.

Iterative BT. For integrating declarative approaches to PMC enumeration into the BT framework, we employ an iterative BT variant that alternates between two phases. The critical observation here is that the dynamic programming of the second phase determines $\mathrm{TW}(G, \Pi')$ for each subset $\Pi' \subset \Pi(G)$. Hence only PMCs of size bounded by the treewidth are required. In particular, if $\mathrm{TW}(G) \leq k-1$, then $\mathrm{TW}(G, \Pi_k(G)) = \mathrm{TW}(G)$. Thus BT-DP requires only the enumeration of $\Pi_k(G)$, i.e., the PMCs with at most k vertices, to determine the treewidth of G. This suggests an iterative variant of BT that alternates for different values of k between (i) ENUM-PMC(G, k), consisting of computing $\Pi_k(G)$ and (ii) BT-DP$(G, \Pi_k(G))$, which either returns ∞, indicating that $\mathrm{TW}(G) \geq k$, or returns the treewidth of G. The original BT algorithm fits this formalism by setting $k = |V(G)|$ during the first iteration, enumerating all PMCs and terminating after the first call to BT-DP. Using declarative approaches for PMC enumeration, we will start with $k = 1$, and increment k after each iteration in order to enumerate the least number of PMCs needed for determining $\mathrm{TW}(G)$.

4 Constraint-Based Enumeration of PMCs

We propose three declarative approaches to enumerating PMCs using SAT and ASP solvers: direct SAT and ASP encodings of the size-k PMCs of a given graph G, which allow for using a SAT/ASP solver to enumerate PMCs, as well as a lazy SAT-based enumeration approach. Each approach can be integrated into the iterative BT algorithm as a substitute for the ENUM-PMC component.

The declarative encodings consist of two parts: (i) encoding of the two conditions of Corollary 1 in terms of connectivity outside of a PMC, and (ii) enforcing that only PMCs of size k are enumerated. Central to the approaches is how the connectivity constraints are encoded in part (i). The encoding of the connectivity constraints is further split to two directions: the *"if"-direction*, and the *"only if"-direction*. In particular, the declarative approaches we propose differ in terms of how the "only if"-direction is handled. The *direct SAT approach* encodes the existence of paths of length m inductively based the existence of paths of length $m-1$. The *direct ASP approach* essentially differs from this by not having to explicate path-lengths. The *lazy SAT approach* uses minimal separators of G to prove that two vertices are not connected outside of a PMC. As the number of minimal separators of a graph can be large, the minimal separator constraints are added *on demand* instead of up front.

For discussing the encodings, let G be a graph with n vertices, and k the size of the PMCs to be enumerated. We assume w.l.o.g that G is connected. We assume familiarity with propositional logic, CNF representations, and answer set programming.

4.1 Direct SAT Encoding of PMCs

We use Boolean variables P_i for $i = 1, \ldots, n$ to indicate the inclusion of vertex v_i in a PMC, and C_{ij} for $i, j = 1, \ldots, n$ as auxiliary variables for encoding the connectivity constraints; for a model τ of the SAT encoding, we will have $\Omega^\tau = \{v_i \mid \tau(P_i) = 1\}$, and $\tau(C_{ij}) = 1$ iff vertices v_i and v_j are connected outside of Ω^τ.

Encoding Corollary 1. The P_i and C_{ij} variables are bound together with the following constraints.

$$\bigwedge_{i=1}^{n} \neg P_i \rightarrow \bigvee_{j=1}^{n} (P_j \wedge \neg C_{ij}) \tag{1}$$

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (P_i \wedge P_j) \rightarrow C_{ij} \tag{2}$$

Constraint 1 encodes Condition (i) of Corollary 1, enforcing that for every vertex $v_i \notin \Omega^\tau$ there is a vertex $v_j \in \Omega^\tau$ such that v_i and v_j are not connected outside of Ω^τ. Constraint 2 encodes Condition (ii), enforcing that if v_i and v_j are in Ω^τ, then they are connected outside of Ω^τ.

"If"-direction of the connectivity constraints. What remains is the encoding of the "if" and "only-if" directions of the connectivity constraints over the C_{ij} variables. The "if"-direction is naturally enforced as follows.

$$\bigwedge_{\{v_i,v_j\}\in E(G)} C_{ij} \tag{3}$$

$$\bigwedge_{i=1}^{n} \bigwedge_{\{v_k,v_j\}\in E(G)} (C_{ik} \wedge \neg P_k) \rightarrow C_{ij} \tag{4}$$

Here Constraint 3 enforces that if there is an edge between vertices v_i and v_j, then they are connected outside any PMC Ω^τ. Constraint 4 ensures that connectivity information is propagated fully: if v_i and v_k are connected outside of Ω^τ, $v_k \notin \Omega^\tau$, and there is an edge between v_k and v_j, then also v_i and v_j are connected outside of Ω^τ.

"Only if"-direction of the connectivity constraints. Note that Constraint 4 is always satisfied by setting $\tau(C_{ij}) = 1$ for all C_{ij}, which is why also the "only if"-direction needs to be enforced explicitly in the direct SAT encoding. This is done using auxiliary variables $C[t]_{ij}$ for $t = 1, \ldots, n$ indicating the existence of a path of length at most t between v_i and v_j that does not containing vertices in Ω^τ. These variables are defined by the following constraints.

$$\bigwedge_{t=1}^{n} \bigwedge_{\{v_i,v_j\}\in E(G)} C[t]_{ij} \tag{5}$$

$$\bigwedge_{\{v_i,v_j\}\notin E(G)} \neg C[1]_{ij} \tag{6}$$

$$\bigwedge_{t=2}^{n} \bigwedge_{\{v_i,v_j\}\notin E(G)} C[t]_{ij} \rightarrow \bigvee_{\{v_k,v_j\}\in E(G)} (C[t-1]_{ik} \wedge \neg P_k) \tag{7}$$

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} C_{ij} \rightarrow C[n]_{ij} \tag{8}$$

Constraints 5 and 6 define the base conditions for being (not) connected. Constraint 7 enforces that if vertices without a direct edge between are connected, there must be a shorter path that supports this connectivity. Constraint 8 binds the path-length variables and the connectivity variables C_{ij}. The redundant constraint $C_{ij}[t] \rightarrow C_{ij}[t+1]$ for all $t = 1..n-1$ and $i, j = 1..n$ is included for additional propagation.

Restricting the size of enumerated PMCs. The restriction to size-k PMCs is enforced through the cardinality constraint

$$\sum_{i=1}^{n} \tau(P_i) = k. \tag{9}$$

For encoding Eq. 9 in CNF, here we use the Totalizer encoding [Bailleux and Boufkhad, 2003] which is both compact and can be used incrementally.

Enumeration of models of the direct SAT encoding, consisting of constraints 1–9, corresponding to the set of size-k PMCs of G, is done in a standard way by incrementally calling a SAT solver, and adding the clause $\bigvee_{\tau(P_i)=1} \neg P_i \vee \bigvee_{\tau(P_i)=0} P_i$ that blocks out each found model τ from further consideration until the SAT solver reports unsatisfiability.

4.2 Direct ASP Encoding of PMCs

As an alternative to the direct SAT encoding we propose a direct ASP encoding. The potential benefits of employing ASP come in terms of the answer set semantics [Gelfond and Lifschitz, 1988], which circumvents—in contrast to the SAT encoding—the need to explicate the lengths of paths in enforcing that the existence of a path must be supported by the existence edges forming the path. The answer set semantics requires natively that all true atoms must have an external support, and the semantics is enforced internally within the ASP solver during SAT-solver like complete search. Furthermore, ASP natively supports cardinality constraints, and ASP solvers offer built-in support for model enumeration.

The ASP encoding analogous to the direct SAT encoding is as follows. The predicate `vertex/1` represents the $N = n$ vertices (`r1`), and `pmc/1` the inclusion of vertices in a PMC; each vertex may be included in a PMC through the choice rule (`r2`), which also enforces the size of the PMC to be k using built-in support for cardinality constraints.

```
vertex(X) :- X = 1..N, vertices(N).          (r1)
{ pmc(X) : vertex(X) } = K :- size(K).       (r2)
```

The input predicate `edge/2` represents the input graph, and `c/2` represents connectivity in the graph (in analogy with the direct SAT encoding) with the following rules `r3`-`r6`.

```
edge(X,Y) :- edge(Y,X).                      (r3)
c(X,X) :- vertex(X).                         (r4)
c(X,Y) :- edge(X,Y).                         (r5)
c(X,Y) :- c(X,Z), not pmc(Z), edge(Z,Y).     (r6)
```

Finally, condition (i) of Corollary 1 is enforced through the integrity constraint

```
:- not pmc(X), 0 = #count{vertex(Y) : d(X,Y)},
               X = 1..N, vertices(N).        (r7)
```

where

```
d(X,Y) :- vertex(X), not c(X,Y), pmc(Y).     (r8)
```

and condition (ii) through

```
:- pmc(X), pmc(Y), not c(X,Y).               (r9)
```

4.3 SAT-based Lazy Approach

As a third alternative declarative approach to handling the connectivity constraints, we propose a lazy SAT approach. The approach is based on enforcing the "only-if"-direction of connectivity constraints through the minimal separators of G. Specifically, v_i and v_j are only connected outside of Ω^τ if Ω^τ does not contain a minimal v_i, v_j separator.

For a fixed minimal separator $S \in \Delta(G)$, we use an auxiliary variable M_S to indicate that S is a subset of Ω^τ. If we were given the set *all* minimal separators, the above condition

can then be enforced by adding the following constraints for each $S \in \Delta(G)$.

$$\left(\bigwedge_{v_i \in S} P_i\right) \rightarrow M_S \tag{10}$$

$$\bigwedge_{i,j:\ S \text{ separates } v_i, v_j} M_S \rightarrow \neg C_{ij} \tag{11}$$

Constraint 10 "activates" the minimal separator if it is a subset of Ω^τ, and Constraint 11 enforces that v_i and v_j are not connected outside of Ω^τ if a minimal separator separating them is activated. In the following we denote by $\text{SEP}(S)$ these constraints for a minimal separator S.

As the number of minimal separators of a graph can be exponential in the number of vertices of G (recall Sect. 3), we do not introduce these constraints for all minimal separators up front. Instead, we propose a lazy approach which iteratively adds these constraints whenever the condition to be enforced is violated by a solution reported by a SAT solver, in which case we can easily obtain a minimal separator for ruling out the unwanted candidate solution from further consideration. The initial number of variables and clauses is $O(n^2)$ and $O(nm)$ in the lazy approach, compared to $O(n^3)$ and $O(n^2 m)$ in the direct approach.

Algorithm 1 details the lazy SAT-based approach to PMC enumeration. A working formula F is maintained, initialized as the conjunction of Constraints 1–4 and 9 (i.e., not including the "only if' encoding of the connectivity constraints). When a model τ of the working formula is obtained via SAT-SOLVE, we check if Ω^τ is an actual PMC of G. This check is straightforward to implement based Corollary 1 and has negligible runtime also in practice. In the positive case, Ω^τ is added to Π_k and blocked from being rediscovered by adding a blocking clause to F. Otherwise, constraint $\text{SEP}(S^\tau)$ is added to F for a minimal separator S^τ to rule out τ as a model for F. Specifically, since F contains constraints defining that Ω^τ is a PMC except the "only if"-direction of connectivity constraints, the fact that Ω^τ is not a PMC implies that there is a pair v_i and v_j such that $\tau(C_{ij}) = 1$ even though v_i and v_j are not connected outside of Ω^τ. The function $\text{VIOLATING-PAIR}(G, \tau)$ returns such a pair of vertices. Next a minimal $\{v_i, v_j\}$-separator $S^\tau \subset \Omega^\tau$ is computed by starting from $S^\tau = \Omega^\tau \setminus \{v_i, v_j\}$ and iteratively removing all vertices $u \in S^\tau$ for which $S^\tau \setminus \{u\}$ still separates v_i, v_j, i.e. $\text{SEPARATES}(S^\tau \setminus \{u\}, G, v_i, v_j)$ is true (the existence of such S^τ follows from v_i and v_j not being connected outside of Ω^τ). Finally, τ is ruled out as a model of F by adding the constraints $\text{SEP}(S^\tau)$. (In addition to τ, the constraints $\text{SEP}(S^\tau)$ may also rule out other models of F that do not correspond to actual PMCs of G.) The function SEPARATES is easy to implement with standard graph traversal algorithms. Algorithm 1 iterates until F becomes unsatisfiable, at which point all PMCs of G of size k have been found.

Algorithm 1 SAT-based lazy enumeration of size-k PMCs

```
F ← conjunction of constraints 1–4 and 9
Π_k ← {}
while τ ← SAT-SOLVE(F) do
  if IS-PMC(Ω^τ) then
    Π_k ← Π_k ∪ {Ω^τ}
    F ← F ∧ (⋁_{τ(P_i)=1} ¬P_i ∨ ⋁_{τ(P_i)=0} P_i)
  else
    v_i, v_j ← VIOLATING-PAIR(G, τ)
    S^τ ← Ω^τ \ {v_i, v_j}
    for all u ∈ S^τ do
      if SEPARATES(S^τ \ {u}, G, v_i, v_j) then
        S^τ ← S^τ \ {u}
    F ← F ∧ SEP(S^τ)
return Π_k
```

5 Adaptations to Generalized Hypertreewidth

While we have so far described the BT framework and the declarative approaches to PMC enumeration in the context of treewidth, adapting them to determining generalized hypertreewidth of a given hypergraph $\mathcal{G}$ requires only minor changes [Moll *et al.*, 2012]; an implementation of BT for GHTW is already available [Korhonen *et al.*, 2019]. In detail, the iterative BT algorithm is adapted for computing $\text{GHTW}(\mathcal{G})$ essentially by making the set of PMCs to be enumerated during iteration k to be the PMCs Ω of $\text{PRIM}(\mathcal{G})$ for which $\text{COV}_\mathcal{G}(\Omega) \leq k$. The edge covers are computed with a branch-and-bound set cover algorithm. In the declarative approaches, this change corresponds to altering the size-k constraint to restrict $\text{COV}_\mathcal{G}(\Omega^\tau)$ instead of restricting $|\Omega^\tau|$. Concretely, in the SAT-based approaches this is done by replacing Constraint 9 by the constraints

$$\bigwedge_{v_i \in V(\mathcal{G})} P_i \rightarrow \bigvee_{v_i \in e} O_e \tag{12}$$

$$\sum_{e \in E(\mathcal{G})} \tau(O_e) = k, \tag{13}$$

where the auxiliary variables O_e for each hyperedge $e \in E(\mathcal{G})$ indicates inclusion of the edge in the edge cover of a PMC. Constraint 12 enforces that a chosen set of edges forms an edge cover, i.e., that the edges cover all $v_i \in \Omega^\tau$. The cardinality constraint 13 limits the size of the smallest edge cover to $\text{COV}_\mathcal{G}(\Omega) \leq k$; in our implementation we again use the Totalizer encoding.

As for the ASP encoding, rule `r2` is replaced by

```
hyper(X) :- X = 1..N, hyperedges(N).
{ selected(X) : hyper(X) } = K :- size(K).
{ pmc(X) : vertex(X) }.
covered(X) :- inhyper(X,Y), selected(Y).
```

to select the hyper edges to the edge cover (`selected/1`) and to determine the covered vertices (`covered/1`) via the input predicate `inhyper/2` representing the end-points of hyperedges. Further, an actual edge cover is enforced with the integrity constraint `:- not covered(X), pmc(X).`

6 Experiments

We integrated the three declarative approaches to PMC enumeration to the recent BT implementation *Triangulator* [Korhonen *et al.*, 2019] that supports both treewidth and generalized hypertreewidth. The implementation is available at

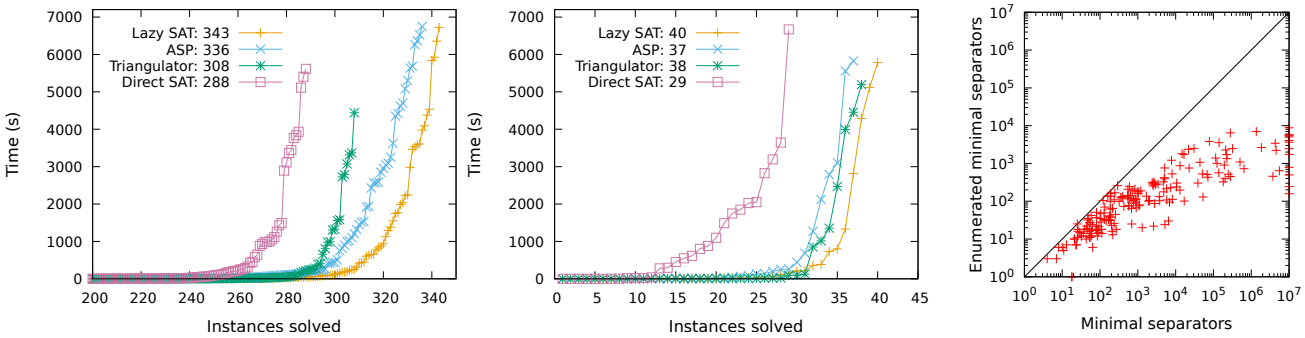

Figure 3: Runtimes for treewidth (left) and generalized hypertreewidth (middle). Right: # separators enumerated by the lazy approach.

https://github.com/Laakeri/pmcenum-ijcai. Triangulator implements PMC enumeration with the original algorithm [Bouchitté and Todinca, 2002] (recall Sect. 3). This allows us to evaluate the effectiveness of the hybrid approaches resulting from replacing the PMC enumeration of Triangulator with the declarative approaches. We made no modifications to the dynamic programming and preprocessing [Bodlaender and Koster, 2006] implemented in Triangulator.

In the experiments, we used Clingo 5.3.0 [Gebser *et al.*, 2016] as the de-facto ASP solver, and—based on preliminary experiments of several state-of-the-art SAT solvers—Glucose 4.1 as the SAT solver [Audemard and Simon, 2018] through its incremental API. For ASP, we used Clingo's built-in support for enumerating all answer sets. All experiments were run single-threaded on computing nodes with 2.4-GHz Intel Xeon E5-2680-v4 processors. A per-instance 2-hour time limit and 32-GB memory limit was imposed.

For empirical corroboration for the claim that the declarative approach circumvents worst cases of the original PMC enumeration within BT, we generated melon graphs (recall Sect. 3) for increasing n. (External preprocessing implemented in Triangulator was disabled for this experiment, as it would determine $\mathrm{TW}(\textsc{Melon}_n) = 2$ for all $n \geq 2$ without BT. Constructions with non-trivial treewidth exhibiting similar behavior may exist.) For treewidth, the original PMC enumeration did not scale beyond $n = 10$ within 2 h, while our lazy SAT-based enumeration allows for determining treewidth for $n = 200$ within 7.5 minutes; see Table 1

For further experiments, we used all of the benchmarks instances from [Korhonen *et al.*, 2019], consisting of 589 graphs for treewidth and 265 for generalized hypertreewidth

n	Triangulator (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

Table 1: Runtime on melon graphs with $3n + 2$ vertices.

gathered from various different sources (including PACE 2016 and 2017 instances). The results are shown in Figure 3 (left) for treewidth and (middle) for generalized hypertreewidth, as the number of instances solved for each of the approaches (x-axis) under different time limits (y-axis). The direct SAT enumeration is not competitive with the other approaches. In contrast, the lazy SAT approach yields best performance, solving 343 treewidth instances compared to 308 solved using Triangulator's original PMC enumeration implementation, also yielding slight improvement for GHTW. ASP enumeration also improves on Triangulator for treewidth, although with some overhead compared to lazy SAT. The largest graph for which treewidth could be determined by the direct SAT, ASP, and lazy SAT methods has 125, 559, and 559 nodes, respectively. For treewidth instances solved with lazy SAT approach, the maximum and median numbers of SAT solver iterations were 157555 and 655. Figure 3 (right) further motivates the lazy SAT approach with a per-instance comparison of the number of minimal separators enumerated by it and the total number of minimal separators. Furthermore, the lazy (direct) approach could decide "treewidth $\leq k$?" for $k = 3, 5, 10, 20$ on 570 (475), 557 (444), 482 (390), 426 (350) instances, respectively. Overall, the results show that declarative PMC enumeration is a promising alternative in the context of BT. In comparison to other state-of-the-art approaches, the PIDDT [Tamaki, 2017] and Jdrasil [Bannach *et al.*, 2017] approaches to treewidth solved 467 and 457 instances, respectively, and the fraSMT [Fichte *et al.*, 2018] approach to GHTW solved 25 instances.

7 Conclusions

We proposed the use of declarative solvers for PMC enumeration within the BT approach to treewidth and generalized hypertreewidth, developing direct SAT and ASP encodings, and as a lazy SAT-based approach for PMC enumeration that circumvents particular exponential behavior of the original PMC enumeration approach for BT. Empirically, declarative PMC enumeration is a promising alternative within BT, improving its performance on treewidth and generalized hypertreewidth.

Acknowledgements

This work was financially supported by Academy of Finland grants 276412 and 312662.

References

[Arnborg *et al.*, 1987] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Discr. Meth.*, 8(2):277–284, 1987.

[Audemard and Simon, 2018] Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27(1):1–25, 2018.

[Bailleux and Boufkhad, 2003] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *CP*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.

[Bannach *et al.*, 2017] Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A Modular Library for Computing Tree Decompositions. In *SEA*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[Berg and Järvisalo, 2014] Jeremias Berg and Matti Järvisalo. SAT-based approaches to treewidth computation: An evaluation. In *ICTAI*, pages 328–335. IEEE, 2014.

[Berry *et al.*, 1999] Anne Berry, Jean-Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. In *WG*, volume 1665 of *LNCS*, pages 167–172. Springer, 1999.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *FAIA*. IOS Press, 2009.

[Bodlaender and Koster, 2006] Hans L. Bodlaender and Arie M.C.A. Koster. Safe separators for treewidth. *Discrete Mathematics*, 306(3):337–350, 2006.

[Bodlaender and Koster, 2008] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.

[Bodlaender *et al.*, 2006] Hans L. Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation. Technical Report UU-CS-2016-052, Utrecht University, 2006.

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

[Bouchitté and Todinca, 2002] Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1):17–32, 2002.

[Darwiche, 2003] Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, 2003.

[Dechter, 2006] Rina Dechter. Tractable structures for constraint satisfaction problems. In *Handbook of Constraint Programming*, pages 209–244. Elsevier, 2006.

[Dell *et al.*, 2018] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The second iteration. In *IPEC 2017*, LIPIcs, pages 30:1–30:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[Fichte *et al.*, 2018] Johannes Klaus Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An SMT approach to fractional hypertree width. In John N. Hooker, editor, *CP*, volume 11008 of *LNCS*, pages 109–127. Springer, 2018.

[Fomin and Villanger, 2008] Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. In *ICALP*, volume 5125 of *LNCS*, pages 210–221. Springer, 2008.

[Fomin *et al.*, 2008] 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.

[Freuder, 1985] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, 1985.

[Ganian *et al.*, 2019] Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-encodings for treecut width and treedepth. In *ALENEX*, pages 117–129. SIAM, 2019.

[Gebser *et al.*, 2016] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with Clingo 5. In *Tech. Commun. ICLP*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

[Gogate and Dechter, 2004] Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In *UAI*, pages 201–208. AUAI Press, 2004.

[Gottlob *et al.*, 2002] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[Gottlob *et al.*, 2009] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32, 2009.

[Gottlob *et al.*, 2010] Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.

[Grohe and Marx, 2014] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1), 2014.

[Korhonen *et al.*, 2019] 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.

[Lauritzen and Spiegelhalter, 1988] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Roy. Statist. Soc. Ser. B*, 5(2):157–224, 1988.

[Lodha *et al.*, 2016] Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. A SAT approach to branchwidth. In *SAT*, volume 9710 of *LNCS*, pages 179–195. Springer, 2016.

[Lodha *et al.*, 2017] Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-encodings for special treewidth and pathwidth. In *SAT*, volume 10491 of *LNCS*, pages 429–445. Springer, 2017.

[Moll *et al.*, 2012] Lukas Moll, Siamak Tazari, and Marc Thurley. Computing hypergraph width measures exactly. *Information Processing Letters*, 112(6):238–242, 2012.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.

[Samer and Veith, 2009] Marko Samer and Helmut Veith. Encoding treewidth into SAT. In *SAT*, volume 5584 of *LNCS*, pages 45–50. Springer, 2009.

[Tamaki, 2017] 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.