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 $N(v)$ of a vertex $v \in V(G)$ consists of vertices $u$ with $\{u, v\} \in E(G)$. For a set $S \subseteq V(G)$, $N(S) = \cup_{v \in S} N(v) \setminus S$. 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 \( E(H) \subset E(H) \). 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 \( \text{MT}(G) \).

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

The width \( W(H) \) of a triangulation \( H \in \text{MT}(G) \) of \( G \) is \( \max_{\omega \in \text{MC}(H)} (|\omega| - 1) \), i.e., the size of the largest clique of \( H \) minus one. The treewidth \( \text{TW}(G) \) of a graph \( G \) is \( \min_{H \in \text{MT}(G)} \{ W(H) \} \), 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 \( H \in \text{MT}(G) \) such that \( \Omega \subset \text{MC}(H) \). 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 \( \text{MT}(G, \Pi_s) \) the minimal triangulations \( H \) of \( G \) for which \( \text{MC}(H) \subset \Pi_s \); and by \( \text{TW}(G, \Pi_s) \) the minimum width of all triangulations in \( \text{MT}(G, \Pi_s) \); \( \text{TW}(G, \Pi_s) = \infty \) if \( \text{MT}(G, \Pi_s) = \emptyset \). Note that \( \text{MT}(G, \Pi_s) = \text{MT}(G) \), \( \text{TW}(G, \Pi_s) = 2 \), and \( \text{TW}(G) = \text{TW}(G, \Pi_n(G)) \leq k - 1 \) if \( \text{TW}(G, \Pi_k(G)) < \infty \).

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 \( H \in \text{MT}(G) \) 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 \( H \in \text{MT}(G, \Pi_3(G)) \) and \( W(H) = 2 \).

We denote the set of connected components of \( G \) by \( \text{CC}(G) \).

For a \( S \subset V(G) \), a component \( C \in \text{CC}(G \setminus S) \) is full wrt \( S \) if \( N(C) \subset S \); 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 \( S' \subset S \) 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 \( N(C) \) into a clique for each \( C \in \text{CC}(G \setminus K) \) makes \( K \) a clique.

![Figure 1](image-url) An example graph (left) and one of its triangulations (right)
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 $TW(G, \Pi(G))$ into the computation of $TW(G[S \cup C], \Pi(G))$ for minimal separators $S \in \Delta(G)$ and full components $C \in C(G \setminus S)$ [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 Figure 2 for $\text{MELON}_{50}$. 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 impractical 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 $TW(G, \Pi')$ for each subset $\Pi' \subset \Pi(G)$. Hence only PMCs of size bounded by the treewidth are required. In particular, if $TW(G) \leq k - 1$, then $TW(G, \Pi_k(G)) = TW(G)$. 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 $\infty$, indicating that $TW(G) \geq k$, 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 $TW(G)$.

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 explicite 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 $\tau$ 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 $\Omega^\tau$.

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

\[
\bigwedge_{i=1}^{n} \neg P_i \rightarrow \bigwedge_{j=1}^{n} (P_j \land \neg C_{ij}) \tag{1}
\]

\[
\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (P_i \land P_j) \rightarrow C_{ij} \tag{2}
\]
Constraint 1 encodes Condition (i) of Corollary 1, enforcing that for every vertex \( v_i \notin \Omega^* \) there is a vertex \( v_j \in \Omega^* \) such that \( v_i \) and \( v_j \) are not connected outside of \( \Omega^* \). Constraint 2 encodes Condition (ii), enforcing that if \( v_i \) and \( v_j \) are in \( \Omega^* \), then they are connected outside of \( \Omega^* \).

**"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.

\[
\bigwedge_{i=1}^{n} \bigwedge_{(v_i, v_j) \in E(G)} C_{ij} \tag{3}
\]

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

**“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 \( \Omega^* \). These variables are defined by the following constraints.

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

\[
\bigwedge_{t=2}^{n} \bigwedge_{(v_i, v_j) \notin E(G)} \neg C[t]_{ij} \tag{6}
\]

\[
\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} C_{ij} \rightarrow \bigvee_{(v_i, v_j) \in E(G)} (C[t - 1]_{ik} \land \neg P_k) \tag{7}
\]

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 \( \bigvee_{\tau(P_i) = 0} P_i \lor \bigvee_{\tau(P_i) = k} P_i \) that blocks out each found model \( P \) 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 cardinality constraints.

The ASP encoding analogous to the direct SAT encoding is as follows. The predicate vertex/1 represents the \( N = n \) vertices \( \{v_1, \ldots, v_n\} \) of the input graph, and pmc/3 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.

\[
\text{vertex}(X) :- X = 1..N, \text{vertices}(N). \quad (r1)
\]

\[
\text{pmc}(X) : \text{vertex}(X) \} = K : \text{size}(X). \quad (r2)
\]

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 \((r3)-(r6)\).

\[
\text{edge}(X,Y) :- \text{edge}(Y,X). \quad (r3)
\]

\[
\text{c}(X,Y) :- \text{vertex}(X). \quad (r4)
\]

\[
\text{c}(X,Y) :- \text{edge}(X,Y). \quad (r5)
\]

\[
\text{c}(X,Y) :- \text{c}(X,Z), \text{not pmc}(Z), \text{edge}(Z,Y). \quad (r6)
\]

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

\[
\text{:- pmc}(X), \text{not vertex}(Y), \text{d}(X,Y), \text{c}(X,Y). \quad (r7)
\]

and condition (ii) through

\[
\text{:- pmc}(X), \text{pmc}(Y), \text{not c}(X,Y). \quad (r9)
\]

### 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 \( \Omega^* \) if \( \Omega^* \) does not contain a minimal \( v_i, v_j \) separator.

For a fixed minimal separator \( S \in \Delta(G) \), we use an auxiliary variable \( MS \) to indicate that \( S \) is a subset of \( \Omega^* \). If we were given the set \( all \) minimal separators, the above condition
can then be enforced by adding the following constraints for each \( S \in \Delta(G) \).

\[
\left( \bigwedge_{v_i \in S} P_i \right) \rightarrow M_S
\]

(10)

Constraint 10 "activates" the minimal separator if it is a subset of \( \Omega^r \), and Constraint 11 enforces that \( v_i \) and \( v_j \) are not connected outside of \( \Omega^r \) 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 \( O(n^2m) \) 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 \( \tau \) of the working formula is obtained via \text{SAT-SOLVE}, we check if \( \Omega^r \) 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, \( \Omega^r \) is added to \( \Pi_k \) and blocked from being re-discovered by adding a blocking clause to \( F \). Otherwise, constraint \( \text{SEP}(S^r) \) is added to \( F \) for a minimal separator \( S^r \) to rule out \( \tau \) as a model for \( F \). Specifically, since \( F \) contains constraints defining that \( \Omega^r \) is a PMC except the "only if"-direction of connectivity constraints, the fact that \( \Omega^r \) 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 \( \Omega^r \). The function \text{VIOLATING-PAIR}(G, \tau) \) returns such a pair of vertices.

Next a minimal \( \{ v_i, v_j \} \)-separator \( S^r \subset \Omega^r \) is computed by starting from \( \Omega^r = \Omega^r \setminus \{ v_i, v_j \} \) and iteratively removing all vertices \( u \in S^r \) for which \( S^r \setminus \{ u \} \) still separates \( v_i, v_j \), i.e., \( \text{SEPARATES}(S^r \setminus \{ u \}, G, v_i, v_j) \) is true (the existence of such \( S^r \) follows from \( v_i \) and \( v_j \) not being connected outside of \( \Omega^r \)). Finally, \( \tau \) is ruled out as a model of \( F \) by adding the constraints \( \text{SEP}(S^r) \). (In addition to \( \tau \), the constraints \( \text{SEP}(S^r) \) may also rule out other models of \( F \) that do not correspond to actual PMCs of \( G \).) The function \text{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

\[
F \leftarrow \text{conjunction of constraints 1–4 and 9}
\]

\[
\Pi_k \leftarrow \{ \}
\]

\[
\text{while } \tau \leftarrow \text{SAT-SOLVE}(F) \text{ do}
\]

\[
\text{if } 1 \text{-PMCs}(\Omega^r) \text{ then}
\]

\[
\Pi_k \leftarrow \Pi_k \cup \{ \Omega^r \}
\]

\[
F \leftarrow F \land \left( \lor_{\tau(P_i)=1} \neg P_i \lor \lor_{\tau(P_i)=0} P_i \right)
\]

\[
\text{else}
\]

\[
v_i, v_j \leftarrow \text{VIOLATING-PAIR}(G, \tau)
\]

\[
S^r \leftarrow \Omega^r \setminus \{ v_i, v_j \}
\]

\[
\text{for all } u \in S^r \text{ do}
\]

\[
\text{if } \text{SEPARATES}(S^r \setminus \{ u \}, G, v_i, v_j) \text{ then}
\]

\[
S^r \leftarrow S^r \setminus \{ u \}
\]

\[
F \leftarrow F \land \text{SEP}(S^r)
\]

\[
\text{return } \Pi_k
\]

of treewidth, adapting them to determining generalized hypertreewidth of a given hypergraph \( G \) 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 GHTW(\( G \)) essentially by making the set of PMCs to be enumerated during iteration \( k \) to be the PMCs \( \Omega \) of PRIM(\( G \)) for which \( \text{COV}_G(\Omega) \leq k \). 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 \( \text{COV}_G(\Omega^r) \) instead of restricting \( \Omega^r \). Concretely, in the SAT-based approaches this is done by replacing Constraint 9 by the constraints

\[
\bigwedge_{v_i \in V(G)} P_i \rightarrow \bigvee_{v_i \in E(G)} O_e
\]

(12)

\[
\sum_{e \in E(G)} \tau(O_e) = k
\]

(13)

where the auxiliary variables \( O_e \) for each hyperedge \( e \in E(G) \) 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 \( v_i \in \Omega^r \). The cardinality constraint 13 limits the size of the smallest edge cover to \( \text{COV}_G(\Omega) \leq k \); in our implementation we again use the Totalizer encoding.

As for the ASP encoding, rule \( r2 \) is replaced by

\[
\text{hyper}(X) :- X = 1..N, \text{ hyperedges}(N).
\]

\[
\{ \text{selected}(X) : \text{hyper}(X) \} = K :- \text{size}(K).
\]

\[
\{ \text{pmc}(X) : \text{vertex}(X) \}.
\]

\[
\text{covered}(X) :- \text{inhyper}(X,Y), \text{ selected}(Y).
\]

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 \( :- \text{not covered}(X), \text{pmc}(X) \).

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
https://github.com/Laakeri/pmcenum-ijcai. Triangulator implements PMC enumeration with the original algorithm [Bouchitte 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 TW(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 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.

### Table 1: Runtime on melon graphs with \( 3n + 2 \) vertices.

<table>
<thead>
<tr>
<th>( n )</th>
<th>Triangulator (s)</th>
<th>Lazy SAT (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.46</td>
<td>0.04</td>
</tr>
<tr>
<td>7</td>
<td>3.27</td>
<td>0.06</td>
</tr>
<tr>
<td>8</td>
<td>26.80</td>
<td>0.06</td>
</tr>
<tr>
<td>9</td>
<td>261.20</td>
<td>0.09</td>
</tr>
<tr>
<td>10</td>
<td>2576.73</td>
<td>0.09</td>
</tr>
<tr>
<td>20</td>
<td>TO</td>
<td>0.38</td>
</tr>
<tr>
<td>50</td>
<td>TO</td>
<td>3.52</td>
</tr>
<tr>
<td>100</td>
<td>TO</td>
<td>39.00</td>
</tr>
<tr>
<td>200</td>
<td>TO</td>
<td>384.95</td>
</tr>
</tbody>
</table>

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


