Discrete Applied Mathematics 303 (2021) 216–236

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Efficiently enumerating minimal triangulations

Nofar Carmeli [a,*], Batya Kenig [a], Benny Kimelfeld [a], Markus Kröll [b]

[a] Technion, Haifa, Israel
[b] TU Wien, Vienna, Austria

ARTICLE INFO

Article history:
Received 19 July 2019
Received in revised form 30 April 2020
Accepted 29 May 2020
Available online 22 June 2020

Keywords:
Minimal triangulation
Tree decomposition
Enumeration algorithm
Minimal separators
Maximal independent sets
Maximal cliques

ABSTRACT

We present an algorithm that enumerates all the minimal triangulations of a graph in incremental polynomial time. Consequently, we get an algorithm for enumerating all the proper tree decompositions, in incremental polynomial time, where "proper" means that the tree decomposition cannot be improved by removing or splitting a bag. The algorithm can incorporate any method for (ordinary, single result) triangulation or tree decomposition, and can serve as an anytime algorithm to improve such a method. We describe an extensive experimental study of an implementation on real data from different fields. Our experiments show that the algorithm improves upon central quality measures over the underlying tree decompositions, and is able to produce a large number of high-quality decompositions.

1. Introduction

Many intractable computational problems on graphs admit tractable algorithms when applied to trees or forests. A *tree decomposition* extracts a tree structure from a graph by grouping nodes into *bags* (each treated as a single node). The corresponding operation on hypergraphs is that of a *generalized hypertree decomposition* [20] that consists of a tree decomposition of the *primal* graph (which has the same set of nodes, and an edge between every two hyperedge neighbors), and an assignment of hyperedge labels (edge covers) to the tree nodes [21]. Tree decompositions and generalized hypertree decompositions have a plethora of applications, including optimization of (multi)join queries in databases [20,27,42], containment of database queries, constraint satisfaction problems [30], prediction of RNA secondary structure [45], computation of Nash equilibria in games [20], inference in probabilistic graphical models [32], and weighted model counting [28].

Past research has focused on obtaining a "good" tree decompositions, where goodness is typically defined as having low *tree width* [39]—the maximal cardinality of a bag (minus one). Nevertheless, finding a tree decomposition of the minimal tree width is NP-hard [2], as is the case for other common measures of goodness for tree decompositions such as *fill* [44], and in the case of hypergraphs *hypertree width* [22], *generalized hypertree width* [23], and *fractional hypertree width* [34]. Therefore, heuristic algorithms are often applied [4,6]. The different measures of goodness are motivated by the fact that the needs of different applications are often different from (though related to) the width. Additional examples are the complexity of weighted model counting, induced by a parameter associated with the "CNF-tree" of the formula [21,28], and the effectiveness of *adhesions* (parent–child intersection) for caching in terms of dimension and skew [27]. In fact, Kalinsky et al. [27] have illustrated how, in real-life scenarios, isomorphic tree decompositions of a minimal width may result in orders-of-magnitude difference in (join) performance.

* Corresponding author.
E-mail addresses: snofca@cs.technion.ac.il (N. Carmeli), batyak@cs.technion.ac.il (B. Kenig), bennyk@cs.technion.ac.il (B. Kimelfeld), kroell@dbai.tuwien.ac.at (M. Kröll).

https://doi.org/10.1016/j.dam.2020.05.034
0166-218X/

The common approach is to devise a decomposition algorithm (exact, approximate or heuristic) to capture the desired measure of goodness per application. However, this is a nontrivial challenge that (to the least) requires high expertise in algorithms and tree decompositions. We propose an alternative approach—produce a large number of different tree decompositions, using a baseline decomposition method, and allow the application at hand to choose the best according to its internal measure function. Our approach brings together results and techniques from the areas of *chordal graphs* and *enumeration theory* in order to establish a practical tool for enhancing decomposition algorithms and, by implication, the performance of various inference and optimization algorithms. Specifically, we explore the task of *enumerating* all (or a subset of) the tree decompositions. Such algorithms have been proposed in the past for small graphs (representing database queries), without complexity guarantees [42]. Our main result is an enumeration algorithm that runs in *incremental polynomial time* [25], that is, the time between producing the Nth result and the $(N+1)$st result is polynomial in N and in the size of the input.

We first need to define which tree decompositions should be enumerated, as many of them are effectively useless. For example, if we take a graph that is already a tree, we do not wish to enumerate the tree decompositions that group nodes with no reason; in fact, the tree itself is the only reasonable decomposition in this case. Therefore, we consider only tree decompositions that cannot be "improved" by removing or splitting a bag, and we call such tree decompositions *proper*. We prove that the proper tree decompositions are in a bijective (and efficiently computable) correspondence to the *minimal triangulations* of the graph at hand, defined a follows. A *triangulation* of a graph g is a graph g' that is obtained from g by adding edges so that g' is *chordal*, that is, g' does not have any induced simple cycle of more than three nodes. A triangulation is *minimal* if no triangulation can be obtained using only a strict subset of the added edges.

So, the problem is reduced to the task of enumerating all of the minimal triangulations of a graph. In this manuscript we devise an algorithm for performing this task in incremental polynomial time. Our approach is as follows. Parra and Scheffler [36] have shown that there is a one-to-one correspondence between the minimal triangulations of a graph g and the maximal independent sets of a special graph $\mathcal{G}$. The nodes of $\mathcal{G}$ are the so called *minimal separators of* g, and the edges are between *crossing* minimal separators. (Precise definitions are in Section 2.) So, enumerating the minimal triangulations of a graph boils down to enumerating these maximal sets. It is well known that all the maximal independent sets of a graph can be enumerated with polynomial delay [11,25]. However, this is insufficient for us, since the graph $\mathcal{G}$ is not given as input, and in fact, its number of nodes can be exponential in the size of the original graph g. Therefore, we cannot construct this graph ahead of time, and cannot directly use existing algorithms to establish incremental polynomial time.

We address this problem by defining an abstraction of the graph $\mathcal{G}$ of minimal separators by means of a *Succinct Graph Representation* (SGR), which is represented compactly by two algorithms: one for enumerating the nodes and one for testing whether a given pair of nodes forms an edge. In particular, we can access the nodes of $\mathcal{G}$ through a polynomial-delay iterator, due to a result by Berry et al. [5] (who show how to enumerate the minimal separators of a graph). Applying previous results, we prove that the SGR for the minimal separator graph (i.e., $\mathcal{G}$) meets certain tractability conditions termed *tractable expansion*, which enable the enumeration of its maximal independent sets (i.e., g's minimal triangulations) in incremental polynomial time in the size of the representation (which can be logarithmic in the size of the graph itself).

In summary, we reduce the problem of enumerating the proper tree decompositions to that of enumerating the minimal triangulations, which we reduce to the problem of enumerating the maximal independent sets of an SGR with tractability properties, and we devise an algorithm for the latter task. An important feature of the algorithm is that it can incorporate any black-box procedure for expanding a given independent set into a maximal one. When applied to enumerating the proper tree decompositions, such a procedure can be any off-the-shelf algorithm for minimal triangulation or tree decomposition (e.g., Maximum Cardinality Search [4] and LB-Triang [6]). However, our algorithm executes this procedure on different versions of the original graph, each time with some new edges added. Hence, our algorithm has the potential of using a high-quality decomposition algorithm for producing *many* high-quality decompositions, enabling the user to choose the best one generated according to the specific measures of her use case (may it be width or anything else).

After establishing our algorithm, we describe an experimental study where we have tested the ability of the algorithm to utilize the aforementioned triangulation algorithms. The experimental study covers graphs of a wide range of domains (where tree decomposition is needed for efficient analysis): join queries (from the TPC-H collection), Bayesian networks, Markov Random Fields, grids, and random graphs. We tested the execution time (delay) of the algorithm, its ability to reduce the width or fill (number of edges added to establish chordality), and the number of decompositions of quality (width/fill) the same or better than that of the original off-the-shelf algorithm. The results show that, indeed, our algorithm can effectively enhance the quality of the corresponding decomposition algorithm.

A short version of this manuscript was published in the 2017 Symposium on Principles of Database Systems (PODS). The current version includes all proofs omitted from the short version. In addition, we have added Sections 2.6 and 3.3, where we discuss the possibility of reaching a polynomial delay algorithm solving our problem, and prove that the approach taken here cannot be used to reach such a time bound.

The rest of the manuscript is organized as follows. In Section 2 we give preliminary definitions and notation, and recall basic results from the literature. The SGR framework is presented in Section 3, along with the enumeration algorithm for maximal independent sets. In Section 4 we prove that the graph of minimal separating sets satisfies the tractability requirements needed for the SGR enumeration algorithm, and thereby establish an algorithm for enumerating the minimal triangulations. We show how this algorithm can enumerate the proper tree decompositions in Section 5. Then, the experimental study is presented in Section 6, and we conclude in Section 7.

2. Preliminaries

In this section we give some basic notation and terminology that we use throughout the paper. In addition, we recall some basic theory that we need in this paper.

2.1. Graphs

The graphs in this work are undirected. For a graph g, the set of nodes is denoted by $\mathsf{V}(g)$, and the set of edges (pairs $\{u, v\}$ of distinct nodes) is denoted by $\mathsf{E}(g)$. Let U be a set of nodes of a graph g. We denote by $g_{|U}$ the subgraph of g *induced* by U; that is, $\mathsf{V}(g_{|U}) = U$ and $\mathsf{E}(g_{|U}) = \{\{u, v\} \in \mathsf{E}(g) \mid \{u, v\} \subseteq U\}$. We denote by $g \setminus U$ the graph obtained from g by removing all the nodes in U (along with their incident edges), that is, the graph $g_{|\mathsf{V}(g)\setminus U}$.

Let g be a graph and U a set of nodes of g. We say that U is an *independent set* if it does not contain both endpoints of any edge, and it is a *maximal independent set* if it is an independent set and it is not strictly contained in any other independent set. We denote by $\mathit{MaxInd}(g)$ the set of all the maximal independent sets of g. We say that U is a *clique* (*of* g) if every two nodes of U are connected by an edge, and it is a *maximal clique* (*of* g) if it is a clique that is not strictly contained in any other clique. We denote by $\mathit{MaxClq}(g)$ the set of all the maximal cliques of g. The operation of *saturating* U (*in* g) is that of connecting every non-adjacent pair of nodes in U by a new edge. Hence, if h is obtained from g by saturating U, then U is a clique of h.

2.2. Minimal separators

Let g be a graph, and let S be a subset of $\mathsf{V}(g)$. Let u and v be two nodes of g. We say that S is a (u, v)-*separator* if u and v belong to distinct connected components of $g \setminus S$. We say that S is a *minimal* (u, v)-separator if no strict subset of S is a (u, v)-separator. We say that S is a *minimal separator* if there are two nodes u and v such that S is a minimal (u, v)-separator. We denote by $\mathit{MinSep}(g)$ the set of all the minimal separators of g. We mention that the number of minimal separators (i.e., $|\mathit{MinSep}(g)|$) may be exponential in the number of nodes (i.e., $|\mathsf{V}(g)|$).

Let g be a graph, and let S and T be two minimal separators of g. We say that S *crosses* T, in notation $S \natural_g T$, if there are nodes u and v in T such that S is a (u, v)-separator. If g is clear from the context, we may omit it and write simply $S \natural T$. It is known that $\natural$ is a symmetric relation: if S crosses T then T crosses S [29,36]. Hence, if $S \natural T$ then we may also say that S and T are *crossing*. When S and T are non-crossing, then we also say that S and T are *parallel*.

2.3. Chordality and triangulation

Let g be a graph. A *cycle* of g is a path that starts and ends with the same node. A *chord* of a cycle c of g is an edge $e \in \mathsf{E}(g)$ that connects two nodes that are non-adjacent in c. We say that g is *chordal* if every cycle of length greater than three has a chord. Whether a given graph is chordal can be decided in linear time [41]. Dirac [15] has shown a characterization of chordal graphs by means of their minimal separators.

Theorem 2.1 (*Dirac [15]*)**.** *A graph g is chordal if and only if every minimal separator of g is a clique.*

Rose [40] has shown that a chordal graph g has fewer minimal separators than nodes (that is, if g is chordal then $|\mathit{MinSep}(g)| < |\mathsf{V}(g)|$), and Kumar and Madhavan [31] have shown that we can find all of these minimal separators in linear time.

Theorem 2.2 (*Kumar and Madhavan [31]*)**.** *Let g be a chordal graph. The set $\mathit{MinSep}(g)$ can be computed in linear time.*

A *triangulation* of a graph g is a graph h such that $\mathsf{V}(g) = \mathsf{V}(h)$, $\mathsf{E}(g) \subseteq \mathsf{E}(h)$, and h is chordal. The edges in $\mathsf{E}(h) \setminus \mathsf{E}(g)$ are commonly referred to as *fill edges*. A *minimal triangulation* of g is a triangulation h of g with the following property: for every graph h' with $\mathsf{V}(g) = \mathsf{V}(h')$, if $\mathsf{E}(g) \subseteq \mathsf{E}(h') \subsetneq \mathsf{E}(h)$ then h' is non-chordal (or in other words, h' is not a triangulation of g). In particular, if g is already chordal then g is the only minimal triangulation of itself. We denote by $\mathit{MinTri}(g)$ the set of all the minimal triangulations of g.

2.4. Tree decomposition

Let g be a graph. A *tree decomposition* d of g is a pair (t, β), where t is a tree and $\beta : \mathsf{V}(t) \to 2^{\mathsf{V}(g)}$ is a function that maps every node of t into a set of nodes of g, so that all of the following hold.

- Nodes are covered: for every node $u \in \mathsf{V}(g)$ there is a node $v \in \mathsf{V}(t)$ such that $u \in \beta(v)$.
- Edges are covered: for every edge $e \in \mathsf{E}(g)$ there is a node $v \in \mathsf{V}(t)$ such that $e \subseteq \beta(v)$.
- *Junction-tree* (or *running-intersection*) property: for all nodes $u, v, w \in \mathsf{V}(t)$, if v is on the path between u and w, then $\beta(v)$ contains $\beta(u) \cap \beta(w)$.

Let g be a graph, and let $d = (t, \beta)$ be a tree decomposition of g. For a node v of t, the set $\beta(v)$ is called a *bag* of d. We denote by $\text{bags}(d)$ the set $\{\beta(v) \mid v \in \mathsf{V}(t)\}$, and we denote by $\text{saturate}(g, d)$ the graph obtained from g by saturating (i.e., adding an edge between every pair of nodes in) every bag of d.

Jordan [26] shows the following characterization of chordal graphs by means of tree decompositions.

Theorem 2.3 (*Jordan [26]*)**.** *A graph g is chordal if and only if it has a tree decomposition d such that every bag of d is a clique of g.*

2.5. Enumeration

An *enumeration problem* P is a collection of pairs (x, Y) where x is an *input* and Y is a finite set of *answers* for x, denoted by $\mathsf{P}(x)$. A *solver* for an enumeration problem P is an algorithm that, when given an input x, produces (or *prints*) a sequence of answers such that every answer in $\mathsf{P}(x)$ is printed precisely once. A solver for an enumeration problem is also referred to as an *enumeration algorithm*.

Johnson, Papadimitriou and Yannakakis [25] introduced several different notions of *efficiency* for enumeration algorithms, and we recall these now. Let P be an enumeration problem, and let A be solver for P. We say that A runs in:

- *polynomial total time* if the total execution time of A is polynomial in $(|x| + |\mathsf{P}(x)|)$;
- *polynomial delay* if the time between printing every two consecutive answers is polynomial in $|x|$;
- *incremental polynomial time* if, after printing N answers, the time to print the next $(N + 1)$st answer is polynomial in $(|x| + N)$.[1]

Observe that a solver that enumerates with polynomial delay also enumerates with incremental polynomial time, which, in turn, implies polynomial total time.

2.6. Enumerating the minimal triangulations

A common approach to establish enumeration with polynomial delay is via the technique known as the *branch-and-bound* (or the *flashlight*) method [9]. In this approach, we find a condition ψ over the answers, and then recursively enumerate all of the answers that satisfy ψ and all of the answers that violate ψ (i.e., satisfy $\psi' = \neg\psi$). Hence, in each recursive call, we need to enumerate all the answers that satisfy a conjunction $\psi_1 \wedge \cdots \wedge \psi_m$ of such conditions. For this approach to guarantee polynomial delay, the depth of the recursion should be bounded by a polynomial in the size of the input. Importantly, in each recursive call, we should be able to test whether there is *at least one* answer that satisfies $\psi_1 \wedge \cdots \wedge \psi_m$. Then, in the leaves, we should be able to produce the single answer that satisfies the given constraints.

In this manuscript, we devise an algorithm for enumerating the minimal triangulations: given g, enumerate $\mathit{MinTri}(g)$. A branch-and-bound attempt to solve this problem would be, say, to apply the conditions of inclusion and exclusion of fill edges. This approach amounts to testing whether there is a minimal triangulation that contains a given set of edges and excludes another given set of edges. Unfortunately, it follows from known hardness results of Golumbic, Kaplan and Shamir [19] that this problem is intractable.

Proposition 2.4. *The following decision problem is NP-complete: given g and two sets I and X of node pairs, is there a minimal triangulation h of g such that $I \subseteq \mathsf{E}(h)$ and $\mathsf{E}(h) \cap X = \emptyset$?*

Proof. Membership in NP is straightforward. To show hardness, we use a reduction from the *chordal sandwich problem*. For a graph property Π, the sandwich problem for Π is that of determining, given graphs g and g'' with $\mathsf{V}(g) = \mathsf{V}(g'')$ and $\mathsf{E}(g) \subseteq \mathsf{E}(g'')$, where there exists a graph g' such that $\mathsf{V}(g') = \mathsf{V}(g)$, $\mathsf{E}(g) \subseteq \mathsf{E}(g') \subseteq \mathsf{E}(g'')$, and g' satisfies Π. Golumbic et al. [19] proved the NP-hardness of this problem for various graph properties Π, including chordality. Now, given g and g'', let X be the set of all the node pairs that are *not* edges of g''. The existence of g' in the chordal sandwich problem is equivalent to the existence of a (minimal) triangulation of g that excludes X. □

Hence, we adopt a different approach to enumerating the minimal triangulations, as we describe in the following sections.

3. Enumerating maximal independent sets on succinct graphs

The main result of this paper is an algorithm for enumerating the minimal triangulations of a graph g. As we explain in the next section, this problem amounts to enumerating the maximal independent sets of a graph h. It is known that

[1] The definition of Johnson et al. [25] requires the delay to be polynomial in the size of the input and the *size* of the previously produced results (not just their *number* N as we define here). However, the definitions are equivalent when the size of each answer is polynomial in that of the input, as in our case.

the maximal independent sets of a graph can be enumerated with polynomial delay [25]. However, we cannot instantiate h, since the number of nodes of h can be exponential in the size of g. Hence, known algorithms for enumerating maximal independent sets cannot be applied to solve our problem. Nevertheless, h possesses some tractability properties that, in fact, allow us to efficiently enumerate the maximal independent sets of h. In this section we identify these properties within an abstract framework of *succinct graph representations*, where a graph may be exponentially larger than its representation, and we have access to the nodes and edges through efficient algorithms. Mainly, we devise an algorithm for enumerating the maximal independent sets for such graphs.

3.1. Succinct graph representations

We begin with the formal definition of a succinct graph representation.

Definition 1 (*SGR*)**.** A *Succinct Graph Representation* (*SGR*) is a triple $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$, where:

- $\mathcal{G}$ is a function that maps every input x, referred to as an *instance*, to a graph $\mathcal{G}(x)$;
- A_{V} is an enumeration algorithm that, given an instance x, enumerates the nodes of $\mathcal{G}(x)$;
- A_{E} is a decision algorithm that, given an instance x and nodes v and u of $\mathcal{G}(x)$, determines whether v and u are connected by an edge in $\mathcal{G}(x)$.

An SGR $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$ is said to be *tractably accessible* if both the following hold.

1. A_{V} enumerates with polynomial delay.
2. A_{E} terminates in polynomial time.

Here, both polynomials are with respect to $|x|$ (the length of x). Observe that in a tractably accessible SGR, the (representation) size of every node v of $\mathcal{G}(x)$ is polynomial in that of x (since writing v is within the polynomial delay).

For efficient enumeration of $\mathit{MaxInd}(\mathcal{G}(x))$, we need some more tractability conditions.

Definition 2 (*Tractable Expansion*)**.** A tractably accessible SGR $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$ is said to have a *tractable expansion* if both of the following conditions hold.

1. There is a polynomial p such that $|I| \leq p(|x|)$ for all instances x and independent sets I of $\mathcal{G}(x)$.
2. There is a polynomial-time algorithm that, given x and an independent set I of $\mathcal{G}(x)$, either determines that I is maximal or returns a node $v \notin I$ such that $I \cup \{v\}$ is independent.

Following is an example of an SGR that is central to this paper.

3.1.1. The separator graph as an SGR

The *separator graph* of a graph g is the graph that has the set $\mathit{MinSep}(g)$ of minimal separators as its node set, and an edge between every two minimal separators that are crossing (i.e., $S, T \in \mathit{MinSep}(g)$ such that $S \natural T$). Throughout this paper we denote by MSGraph the SGR $(\mathcal{G}^{\mathsf{ms}}, A_{\mathsf{V}}^{\mathsf{ms}}, A_{\mathsf{E}}^{\mathsf{ms}})$, where:

- $\mathcal{G}^{\mathsf{ms}}$ is a function that maps the representation of a graph g to its separator graph $\mathcal{G}^{\mathsf{ms}}(g)$.
- $A_{\mathsf{V}}^{\mathsf{ms}}$ is an enumeration algorithm that, given a graph g, enumerates its set $\mathit{MinSep}(g)$ of minimal separators. We can use here a variation of the algorithm of Berry et al. [5] that enumerates $\mathit{MinSep}(g)$ with polynomial delay, as we describe later in Section 4.2.
- $A_{\mathsf{E}}^{\mathsf{ms}}$ is an algorithm that, given a graph g and two minimal separators S and T, determines whether $S \natural T$ efficiently (e.g., by removing S and testing whether T is split along multiple connected components).

In particular, MSGraph is a tractably accessible SGR.

3.2. Enumerating maximal independent sets in SGRs

Our main result for this section is the following.

Theorem 3.1. *Let $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$ be a tractably accessible SGR with a tractable expansion. There is an algorithm that, given an instance x, enumerates the set $\mathit{MaxInd}(\mathcal{G}(x))$ in incremental polynomial time.*

The proof is via the algorithm EnumMIS that is depicted in Fig. 1. This algorithm is an adaptation of the algorithm for computing full disjunctions in databases [10] that generalizes the problem of enumerating maximal cliques (or maximal independent sets). In turn, that algorithm was based on an improvement of the algorithm of Lawler et al. [33] for generating the maximal independent sets in polynomial total time, and all rely on the general idea of reducing the problem to the *input-restricted problem* that was later introduced by Cohen et al. [11] for enumerating maximal node sets that satisfy a hereditary property.

Algorithm EnumMIS(x)

```
1:  J := Extend(x, ∅)
2:  print J
3:  Q := {J}
4:  P := ∅
5:  V := ∅
6:  iterator := A_V(x)
7:  while Q ≠ ∅ do
8:    J := Q.pop()
9:    P.push(J)
10:   for all v ∈ V do
11:     J_v := {v} ∪ {u ∈ J | ¬A_E(x, v, u)}
12:     K := Extend(x, J_v)
13:     if K ∉ Q ∪ P then
14:       print K
15:       Q := Q ∪ {K}
16:   while Q = ∅ and iterator.hasNext() do
17:     v := iterator.next()
18:     V := V ∪ {v}
19:     for all J' ∈ P do
20:       J'_v := {v} ∪ {u ∈ J' | ¬A_E(x, v, u)}
21:       K := Extend(x, J'_v)
22:       if K ∉ Q ∪ P then
23:         print K
24:         Q := Q ∪ {K}
```

Fig. 1. Enumerating maximal independent sets of an input x for an SGR $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$.

The underlying idea of that algorithm is to construct a graph over the space of the solutions (maximal independent sets), and traverse the graph in a depth-first-search manner. In the case of maximal independent sets, there is an edge from J to K if K is obtained from J by adding a new node v, removing the neighbors of v, and greedily extending to a maximal independent set.

In this section, we describe the algorithm and prove its correctness and efficiency. In the remainder of this section, we fix a tractably accessible SGR $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$ with tractable expansion, and an input instance x. Our goal is to enumerate $\mathit{MaxInd}(\mathcal{G}(x))$.

3.2.1. Algorithm description

As explained earlier, the algorithm extends every maximal independent set J that it generates in the direction of every node v that it generates. By *extending J in the direction of v* we mean producing an arbitrary maximal independent set K that contains v and all nodes in J that are non-neighbors of v. As long as there are unprocessed sets, they are extended in the direction of all previously generated nodes. When no unprocessed sets are left, additional nodes are generated, and the previously processed sets are extended in the direction of the new nodes. Put differently, our algorithm adapts the traversal approach by restricting the steps to the solutions that are obtained by extending in the direction of the nodes v that have been produced until that point of time; when a new node v is generated, we revisit the past solutions and take the steps implied by v.

The algorithm maintains two collections, $\mathcal{Q}$ and $\mathcal{P}$, for storing answers (which are maximal independent sets of the graph $\mathcal{G}(x)$). The algorithm inserts answers into $\mathcal{Q}$, and repeatedly *removes* (or *pops*) an answer from $\mathcal{Q}$ and *processes* that answer (while possibly inserting new answers into $\mathcal{Q}$), until $\mathcal{Q}$ is empty. The set $\mathcal{P}$ stores the answers that have already been removed from $\mathcal{Q}$ and processed. Importantly, both collections feature membership testing, element removal and element insertion with a number of comparisons logarithmic in their cardinality (i.e., the number of answers they hold at the time of the operation). In addition, the algorithm maintains a collection $\mathcal{V}$ of the nodes of $\mathcal{G}(x)$ generated thus far. The collection $\mathcal{Q}$ is initialized with a single result (which is printed after being generated), which is an arbitrary maximal independent set. This result is obtained through the procedure $\mathsf{Extend}(x, I)$ that extends a given independent set I into a maximal one. Note that this procedure can be implemented in polynomial time, since $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$ has a tractable expansion. The sets $\mathcal{P}$ and $\mathcal{V}$ are initialized empty.

The algorithm accesses the nodes of $\mathcal{G}(x)$ through an iterator object that is obtained by executing $A_{\mathsf{V}}(x)$, and features two polynomial-time operations:

- Boolean hasNext() determines whether there are additional nodes of $\mathcal{G}(x)$ to enumerate.
- next() returns the next node in the iteration.

The algorithm applies the iteration of line 7 until $\mathcal{Q}$ becomes empty, and then terminates. In every iteration, the algorithm pops an element from $\mathcal{Q}$, stores it in $\mathcal{P}$ (lines 8–9), and then processes it. The algorithm iterates through the nodes in $\mathcal{V}$, and for each node v it applies (in lines 11–15) what we call *extension of J in the direction of v*:

1. Generate the set J_v that consists of v and all the nodes in J that are non-neighbors of v, using the algorithm A_{E} for testing adjacency;
2. Extend J_v into an arbitrary maximal independent set K, again using $\mathsf{Extend}(x, J_v)$;
3. If K is in neither $\mathcal{Q}$ nor $\mathcal{P}$ (meaning it was not printed before), then print K and add it to $\mathcal{Q}$.

Observe that J_v is an independent set, and therefore, it is possible to invoke $\mathsf{Extend}(x, J_v)$ with J_v.

Up to this point, the algorithm is very similar to the algorithm of Cohen et al. [10] for computing full disjunctions, except that $\mathcal{V}$ does not hold all nodes but only the nodes generated so far. The twist (and the source of extra challenge in proving correctness and efficiency) is in lines 16–24, where we generate additional nodes and compensate for them being missing in the previous iterations. In these lines, the algorithm tests whether it is the case that $\mathcal{Q}$ is empty and the node iterator has additional nodes to process (line 16). While this is the case, the algorithm repeats the following procedure (lines 17–24): generate the next node using the iterator of $A_{\mathsf{V}}(x)$, add it to $\mathcal{V}$, and extend *every* previously processed result (i.e., the results in $\mathcal{P}$) in the direction of the newly generated node v (as previously described).

3.2.2. Correctness and efficiency

The following lemma states the correctness of the algorithm: the algorithm enumerates *every* element in $\mathit{MaxInd}(\mathcal{G}(x))$, *only* elements in $\mathit{MaxInd}(\mathcal{G}(x))$, and every element is printed exactly once.

Lemma 3.2. $\mathsf{EnumMIS}(x)$ *enumerates* $\mathit{MaxInd}(\mathcal{G}(x))$.

Proof. The algorithm prints only elements that are created by invoking the procedure Extend. Therefore, the algorithm prints only elements in $\mathit{MaxInd}(\mathcal{G}(x))$. The tests of lines 13 and 22 ensure that whenever an element is printed, this element has not been seen before. Hence, no element is printed more than once. It is left to prove that every maximal independent set of $\mathcal{G}(x)$ is printed by the algorithm.

Observe the following. When the algorithm terminates we have $\mathcal{Q} = \emptyset$. Therefore, in the previous iteration the loop of line 16 could only have terminated due to iterator.hasNext() returning false. Therefore, upon termination $\mathcal{V} = \mathsf{V}(\mathcal{G}(g))$.

Suppose, by way of contradiction, that there is some maximal independent set H that is not printed by the algorithm. Let J be a maximal independent set of $\mathcal{G}(x)$, among all the printed ones, that contains a maximal number of elements from H. The set J must exist, since the algorithm prints at least one maximal independent set. Let H_m be the intersection $H \cap J$. Since $H \neq H_m$ (or else H is not maximal), there is at least one node in $H \setminus J$; let v be such a node.

At this point we have established that before the algorithm terminated, *(a)* the node v has been generated, *and (b) J* has been printed. We now branch into two cases, as follows.

1. The set J was inserted into $\mathcal{P}$ before the node v was generated. Immediately after v is generated (in line 17), the set $J_v = \{v\} \cup \{u \in J \mid \neg A_{\mathsf{E}}(x, v, u)\}$ will be constructed (in line 20) and expanded to a maximal independent set K that contains J_v.
2. The node v was generated before J was inserted into $\mathcal{P}$. At the iteration when J is inserted into $\mathcal{P}$, we have $v \in \mathcal{V}$, and so the set $J_v = \{v\} \cup \{u \in J \mid \neg A_{\mathsf{E}}(x, v, u)\}$ will be constructed (in line 11) and expanded to a maximal independent set K that contains J_v.

So, we have established that before the algorithm terminates, the set J_v is generated and expanded to a maximal independent set K that contains J_v. Furthermore, $H_m \cup \{v\} \subseteq J_v$ (since $H_m \subseteq J$, and does not contain any neighbor of v), and therefore $H_m \cup \{v\} \subseteq K$. According to the algorithm, one of the following options must hold: (1) K is inserted into $\mathcal{Q}$, (2) K is already in $\mathcal{Q}$ (3) K was in $\mathcal{Q}$ in the past and is now in $\mathcal{P}$. Since the algorithm prints every maximal independent set that is inserted into $\mathcal{Q}$, we get a contradiction to the maximality of H_m. □

We now prove that the algorithm EnumMIS enumerates with incremental polynomial time. We do so in two steps. We first define an algorithm that is similar to EnumMIS, but with a small twist that makes it easier to prove incremental polynomial time. Then, we prove a general result that will imply that, if the new algorithm enumerates in incremental polynomial time then so does EnumMIS.

The new algorithm is similar to EnumMIS, except that each of the print commands (lines 2, 14 and 23) is replaced with an operation that takes the time of the printing, but is actually void (e.g., printing to `/dev/null` in Unix). Instead, each maximal independent set is printed immediately after being removed from $\mathcal{Q}$ (line 8). Hence, answers are *held* until removed from $\mathcal{Q}$. We refer to the resulting algorithm as EnumMISHold. For theory purposes, it would have been enough to discuss only EnumMISHold, which is easier to analyze. However, since delaying the results is not required to obtain the theoretical guarantees, we also discuss EnumMIS where we print the results as soon as we have them. Next, we prove that EnumMISHold enumerates in incremental polynomial time. Observe that to bound the delay EnumMISHold, we only need to bound the time between two executions of line 8 of EnumMIS.

Lemma 3.3. $\mathsf{EnumMISHold}(x)$ *enumerates with incremental polynomial time.*

Proof. We begin by showing that the size of the node set $\mathcal{V}$ is polynomial in the size of the printed result set $\mathcal{P}$. Whenever a new node v is inserted into $\mathcal{V}$ (line 18), the set $\mathcal{Q}$ is empty. The following calls to Extend (line 21) will generate maximal independent sets containing v. Each of these maximal independent sets is either already in $\mathcal{P}$, or it is inserted into $\mathcal{Q}$ (line 24). Therefore, at the end of the iteration of the main loop in which v was inserted into $\mathcal{V}$, all maximal independent sets in $\mathcal{Q}$ contain v. In the next iteration of the main loop, if such an iteration exists, one of these newly generated independent sets will be printed and inserted into $\mathcal{P}$ (line 9). That is, at the beginning of every iteration of the algorithm (specifically, line 10), every node $v \in \mathcal{V}$ belongs to some maximal independent set that has already been printed (and thus part of $\mathcal{P}$). Since we assume tractable expansion, each independent set in $\mathcal{P}$ contains at most $p(|x|)$ nodes, and we can conclude that $|\mathcal{V}| \leq p(|x|) \cdot |\mathcal{P}|$.

We now bound the time between two executions of line 8 of EnumMIS. Line 9 takes polynomial time in $|x|$ (since there are at most exponentially many independent sets, $(\mathcal{G}, A_{\mathsf{V}}, A_{\mathsf{E}})$ has a tractable expansion, and operations on $\mathcal{P}$ require a logarithmic number of comparisons in the cardinality). The number of iterations of line 10 is at most the size of $\mathcal{V}$, which is polynomial in the number of answers printed so far (due to the above observation). Each operation in that iteration takes time polynomial in $|x|$.

The loop of line 16 repeats (at most) until a node that belongs to none of the printed answers is generated. Hence, the observation that this number is polynomial in the size of the output, along with the tractable expansion, again implies that we iterate a number of times that is polynomial in the number of answers printed so far. The loop of line 19 repeats at most as many times as the number of answers in $\mathcal{P}$, and all of these have been printed before. Besides the loops, each of lines 17–24 takes polynomial time in $|x|$. □

Lemma 3.3 shows that EnumMISHold enumerates with incremental polynomial time. Next, we show the same for EnumMIS. The key point is that every answer is printed in EnumMIS *no later* than it is printed in EnumMISHold. Note that this holds even though the two algorithms do not necessarily enumerate in the same order (as we make no assumptions about the order of removal in $\mathcal{Q}$), since we assume that EnumMISHold spends on void the printing time of EnumMIS. We will prove that this suffices to conclude that if EnumMISHold enumerates in incremental polynomial time, then so does EnumMIS. We prove here a general result. Let P be an enumeration problem, and let A be a solver for P. For input x and answer $y \in \mathsf{P}(x)$, we denote by $\text{time}_{A,x}(y)$ the time in which y is printed. We prove the following theorem.

Theorem 3.4. *Let* P *be an enumeration problem, and let A and B be two solvers for* P. *Suppose that for all instances x and for all answers $y \in \mathsf{P}(x)$ we have $\text{time}_{A,x}(y) \leq \text{time}_{B,x}(y)$. If B enumerates in incremental polynomial time, then so does A.*

Theorem 3.4 is not a vacuous statement, since the order of results may differ between A and B. Furthermore, the corollary no longer holds when substituting "incremental polynomial time" with "polynomial delay". For example, imagine two algorithms that print all subsets of an input set. The first prints a new answer after every two time ticks, while the second prints them after every single time tick, except for the last answer which is printed at the same time in both algorithms. The first algorithm meets the guarantee of polynomial delay, and even though the second algorithm prints every answer no later than the first, the second algorithm does not enumerate in polynomial delay as its delay before the last answer is exponential.

Let P be an enumeration problem, let A be a solver for P, and let x be input for A. If τ is a time tick during the execution of $A(x)$, then we denote by $\text{out}_{A,x}(\tau)$ the answers $y \in \mathsf{P}(x)$ that have been printed before time τ is reached. We have the following lemma.

Lemma 3.5. *Let* P *be an enumeration problem, and A a solver for* P. *The following are equivalent.*

1. *A enumerates in incremental polynomial time.*
2. *There is a polynomial p such that for all input x and time tick τ it holds that*

$$p(|x| + |\text{out}_{A,x}(\tau)|) > \tau.$$

Proof. Denote the time of the Nth result by t_N.

1 ⇒ 2. If A enumerates in incremental polynomial time, there exists a polynomial p_1 such that $t_{N+1} - t_N \leq p_1(|x| + N)$. Without loss of generality, we assume that p_1 is monotone (as every polynomial is upper bounded by some monotone polynomial, and we can replace p_1 with such polynomial). We get the following on the printing time of the Nth result.

$$t_N = \sum_{i=1}^{N} t_i - t_{i-1} \leq \sum_{i=1}^{N} p_1(|x| + i - 1) \leq N \cdot p_1(|x| + N - 1)$$

In this case we get that for any time τ there exists a polynomial p_2 such that the following holds.

$$\tau < t_{|\text{out}_{A,x}(\tau)|+1} \leq (|\text{out}_{A,x}(\tau)| + 1) \cdot p_1(|x| + |\text{out}_{A,x}(\tau)|) \leq p_2(|x| + |\text{out}_{A,x}(\tau)|)$$

2 ⇒ 1. Assume now that $p_3(|x| + |\text{out}_{A,x}(\tau)|) > \tau$ for any time τ. Consider the delay after the Nth answer.

$$t_{N+1} - t_N \leq t_{N+1} < p_3(|x| + N + 1)$$

This shows that there exists a polynomial p_4 such that $t_{N+1} - t_N < p_4(|x| + N)$, meaning that A enumerates in incremental polynomial time. □

We can now prove Theorem 3.4.

Proof. Using the characterization of Lemma 3.5, let p be a polynomial such that for all x and τ we have $p(|x|+|\text{out}_{B,x}(\tau)|) > \tau$. The condition of the theorem implies that at every time tick τ, the set of answers printed by B is a subset of the set of answers printed by A, and therefore, $|\text{out}_{A,x}(\tau)| \geq |\text{out}_{B,x}(\tau)|$. Again since we can assume monotonicity, we conclude that $p(|x| + |\text{out}_{A,x}(\tau)|) > \tau$ as well. We use Lemma 3.5 to conclude that A enumerates in incremental polynomial time. □

Using the algorithms EnumMIS and EnumMISHold as A and B in Theorem 3.4, respectively, the combination with Lemma 3.3 implies that EnumMIS enumerates in incremental polynomial time, as claimed.

3.3. Tightness of the algorithm

In the following, we show that the time bounds that EnumMIS achieves are tight since it is not possible to solve the same problem with polynomial delay under the SETH assumption.

We recall that k-SAT is the satisfiability problem over n variables, where every clause contains at most k literals. SETH states that there is no algorithm for solving k-SAT in $O^*(2^{(1-\varepsilon)n})$ time for a fixed ε and all k, where the O^*-notation omits polynomial factors.

Definition 3 (*The Strong Exponential Time Hypothesis*)**.** For every $\varepsilon > 0$ there exists a k such that k-SAT requires time larger than $2^{(1-\varepsilon)n}$ where n is the number of variables.

Let $(\mathcal{G}, A_V, A_E)$ be the tractable expansion of a tractably accessible SGR. We denote by $\text{SMIS}(\mathcal{G}, \text{A}_\text{V}, \text{A}_\text{E})$ the following enumeration problem: Given an instance x, enumerate all $\mathit{MaxInd}(\mathcal{G}(x))$.

Proposition 3.6. *There exists some tractably accessible SGR with a tractable expansion* $(\mathcal{G}, A_V, A_E)$*, such that* $\text{SMIS}(\mathcal{G}, \text{A}_\text{V}, \text{A}_\text{E})$ *cannot be enumerated with a polynomial delay, assuming the* SETH*.*

Proof. Let $k \geq 3$, and let ϕ be an instance of k-SAT with $\text{var}(\phi) = \{x_1, \ldots, x_n\}$ (for readability, we assume that $n \geq 2$ is even). We will show that a polynomial delay algorithm for enumerating $\mathit{MaxInd}(\mathcal{G}(x))$ will decide satisfiability of ϕ within time $2^{n/2} \cdot \text{poly}(|\phi|)$. This is true for any choice of k, which is not possible assuming the SETH.

We first describe the SGR $(\mathcal{G}, A_V, A_E)$. For any string x that is not a k-SAT formula, $\mathcal{G}(x) = \emptyset$. Otherwise, given a k-SAT instance ϕ, we define $\mathcal{G}(x)$ as follows: The set of vertices represents all possible truth assignments on $\frac{n}{2}$ variables twice, with two additional nodes $\bot_A$ and $\bot_B$. Intuitively, V_A corresponds to all possible truth assignments on the variables $x_1, \ldots, x_{\frac{n}{2}}$, and V_B corresponds to all possible truth assignments on the remaining variables $x_{\frac{n}{2}+1}, \ldots, x_n$. That is,

$$\begin{aligned} V_A &= \{A\} \times \{0,1\}^{\frac{n}{2}} \\ V_B &= \{B\} \times \{0,1\}^{\frac{n}{2}} \\ V(\mathcal{G}(\phi)) &= V_A \cup V_B \cup \{\bot_A\} \cup \{\bot_B\}. \end{aligned}$$

To define the set of edges, we first start with edges between the set V_A and V_B. There is an edge (u, v) for $u \in V_A$, $v \in V_B$ if and only if u and v together encode a truth assignment that does not satisfy ϕ. Moreover, we also add all edges between nodes in V_A, between nodes in V_B and certain connections to the nodes $\bot_A$ and $\bot_B$ as follows:

$$\begin{aligned} E_{unsat} = \{\{u,v\} | \exists a_1, \ldots, a_n \in \{0,1\} \text{ s.t.} \\ u = (A, a_1, \ldots, a_{\frac{n}{2}}) \in V_A, v = (B, a_{\frac{n}{2}+1}, \ldots, a_n) \in V_B \text{ and } \phi(a_1, \ldots, a_n) = \text{false}\}. \\ E(\mathcal{G}(\phi)) = E_{unsat} \cup \{\bot_A, \bot_B\} \cup \{\{u,v\} \mid u, v \in V_A\} \cup \{\{u,v\} \mid u, v \in V_B\} \cup \{\{u, \bot_A\} \mid u \in V_A\} \cup \{\{u, \bot_B\} \mid u \in V_B\} \end{aligned}$$

We first note that this SGR is tractably accessible. Indeed, the set of nodes can be enumerated with a polynomial (even constant) delay, and for any $u, v \in \mathcal{G}(\phi)$, we can check whether $\{u, v\} \in E(\mathcal{G}(\phi))$ in polynomial time, since evaluation of any k-SAT formula can be done within polynomial (or even linear) time. To show that this SGR also has a tractable expansion, we note that the set of maximal independent sets of $\mathcal{G}(\phi)$ is given as the union of the sets I_A, I_B and I_{sat} with

$$\begin{aligned} I_A = \{\{u, \bot_B\} \mid u \in V_A\}, \quad I_B = \{\{u, \bot_A\} \mid u \in V_B\} \text{ and} \\ I_{sat} = \{\{u,v\} | \exists a_1, \ldots, a_n \in \{0,1\} \text{ s.t. } u = (A, a_1, \ldots, a_{\frac{n}{2}}) \in V_A, v = (B, a_{\frac{n}{2}+1}, \ldots, a_n) \in V_B \text{ and } \phi(a_1, \ldots, a_n) = \text{true}\}. \end{aligned}$$

Every maximal independent set of $\mathcal{G}(\phi)$ is of size 2, satisfying the first condition of a tractable expansion. For the second condition, we note that every subset I of $V(\mathcal{G})$ of size one can be extended trivially to a maximal independent set (by adding either $\bot_A$, $\bot_B$, or in case that $I \subset \{\bot_A, \bot_B\}$ some arbitrary element from V_A or V_B), and for any subset of size two, we can check whether I is (maximally) independent within polynomial time.

Note that ϕ is satisfiable if and only if $\mathit{MaxInd}(\mathcal{G}(x))$ contains more than the sets I_A and I_B. Assume that we can enumerate $\mathit{MaxInd}(\mathcal{G}(x))$ with a polynomial delay. We can output $2 \cdot 2^{\frac{n}{2}}$ many solutions within time $2^{\frac{n}{2}} \cdot \mathsf{poly}(|\phi|)$, meaning that we can decide whether there are more than $2 \cdot 2^{\frac{n}{2}}$ many maximal independent sets of $\mathcal{G}(\phi)$ within in the same time bound. Since ϕ is satisfiable iff $\mathcal{G}(\phi)$ has at least $2 \cdot 2^{\frac{n}{2}} + 1$ maximal independent sets, we are done. □

3.4. Note on space usage

We conclude this section with a discussion on the space usage. Note that our algorithm may reach an exponential space as it relies on remembering all past answers to avoid the production of duplications. This cost is already incurred in the enumerators of maximal independent sets that form the basis of our algorithm [10,11,33]. However, several algorithms for enumerating maximal independent sets (and more generally maximal sets w.r.t. different properties) guarantee both polynomial delay and polynomial space, including the *reverse search* [3], the algorithm of Conte et al. [12], and the *proximity search* [13]. However, it is not clear to us how these algorithms can be adapted to enumerating the maximal independent sets of an SGR in a manner that limits the space, given that the set of nodes is not known upfront (and in light of Proposition 3.6). Moreover, note that the exponential space of our algorithm is also required for storing the (possibly exponential number of) past generated nodes of the SGR.

A natural question then remains open: can Theorem 3.1 be improved to require only polynomial space (at least when ignoring the space used by invoking the SGR functions)? Particularly, we leave open the question of whether and how the aforementioned polynomial-space algorithms can be adapted to enumerating the maximal independent sets of an SGR, and whether we can avoid storing all produced nodes. It appears that further assumptions on the SGR are required to this aim, and establishing these assumptions is left as a future direction.

4. Enumerating minimal triangulations

In Section 3.1.1 we introduced MSGraph and claimed that it is an SGR. In this section, we will use known results to reduce the problem of enumerating the minimal triangulations of a graph to the problem of enumerating the maximal independent sets for MSGraph. We will describe how to enumerate the nodes of MSGraph with polynomial delay, concluding that it is in fact an SGR. We will further show that MSGraph has a tractable expansion (Definition 2), and therefore Theorem 3.1 can be applied to conclude that the minimal triangulations can be enumerated in incremental polynomial time.

4.1. Reduction

We use the following notation. Let g be a graph. We denote by $\mathit{ClqMinSep}(g)$ the set of minimal separators S of g, such that S is a clique of g. Let φ be a subset of $\mathit{MinSep}(g)$. We denote by $g_{[\varphi]}$ the graph that results from saturating the minimal separators in φ.

Parra and Scheffler [36] have shown the following connection between minimal triangulations and maximal sets of *pairwise-parallel minimal separators* (that is, every two minimal separators in the set are non-crossing).

Theorem 4.1 (*Parra and Scheffler [36]*)**.** *Let g be a graph.*

1. *If φ is a maximal set of pairwise-parallel minimal separators of g, then $g_{[\varphi]}$ is a minimal triangulation of g, and $\mathit{MinSep}(g_{[\varphi]}) = \varphi$.*
2. *If h is a minimal triangulation of g, then the set $\varphi = \mathit{MinSep}(h)$ is a maximal set of pairwise-parallel minimal separators in g, and $h = g_{[\varphi]}$.*

Theorem 4.1, combined with Theorem 2.2, gives the desired reduction in the following corollary. Recall that the graph $\mathcal{G}^{\mathsf{ms}}(g)$ is defined in Section 3.1.1, as part of the SGR $\mathsf{MSGraph} = (\mathcal{G}^{\mathsf{ms}}, A_{\mathsf{V}}^{\mathsf{ms}}, A_{\mathsf{E}}^{\mathsf{ms}})$.

Corollary 4.2. *For a graph g, there is a polynomial-time-computable bijection between the following two sets:*

- *$\mathit{MaxInd}(\mathcal{G}^{\mathsf{ms}}(g))$, that is, the set of all maximal sets of pairwise-parallel minimal separators of g.*
- *$\mathit{MinTri}(g)$, that is, the set of all minimal triangulations of g.*

Hence, it suffices to prove that MSGraph has a tractable expansion, which we do next.

Algorithm PDelayAllMinSep(g)

```
1:  Q := ∅
2:  P := ∅
3:  for all v ∈ V(g) do
4:    for all C ∈ C({v} ∪ N(v)) do
5:      Q := Q ∪ {N(C)}
6:  while Q ≠ ∅ do
7:    S := Q.pop()
8:    for all x ∈ S do
9:      S' := {N(C) | C ∈ C(S ∪ N(x))}
10:     if S' ∉ P then
11:       Q := Q ∪ {S'}
12:   P := P ∪ {S}
13:   print S
```

Fig. 2. Enumerating *MinSep*(g) with polynomial delay (a variation of the algorithm by Berry et al. [5]).

4.2. Enumerating minimal separators

We now describe a variation of the algorithm of Berry et al. [5] that, given a graph g, enumerates its set *MinSep*(g) of minimal separators. Their algorithm enumerates with polynomial total time, and with a simple change (that we explain next) can enumerate with polynomial delay. Our variation is depicted in Fig. 2. There, for $v \in \mathsf{V}(g)$ we denote by $N(v)$ the set of neighbors of v. For $U \subseteq \mathsf{V}(g)$ we denote by $N(U)$ the set of neighbors of nodes in U, excluding the nodes of U themselves; that is,

$$N(U) \stackrel{\text{def}}{=\!=} \left(\bigcup_{v \in U} N(v)\right) \setminus U.$$

We also denote by $\mathscr{C}(U)$ the set of connected components of the graph $g \setminus U$ (the graph obtained from g by removing all the nodes of U).

The algorithm remains intrinsically the same as that of Berry et al. [5]. Minimal separators are considered as neighborhoods of connected components. The algorithm finds minimal separators contained in a set $U \subseteq \mathsf{V}(g)$ by taking the neighborhoods of the connected components of $g \setminus U$, that is, $N(C)$ for all $C \in \mathscr{C}(U)$. Initially, the minimal separators that are contained in the neighborhoods of single nodes are generated (lines 3–5). Then, every previously generated minimal separator S is processed to produce more minimal separators that are *close* to S (lines 7–12). For every node v in the minimal separator S, it produces minimal separators that are contained in $S \cup N(v)$.

Our modification is in the data structures and the time of printing answers. In Fig. 2, $\mathcal{Q}$ and $\mathcal{P}$ play the role of $\mathscr{S} \setminus \mathscr{T}$ and $\mathscr{T}$ of the original algorithm [5], respectively. There, $\mathscr{S}$ holds all minimal separators generated, and $\mathscr{T}$ is a subset that holds the separators that were processed. The easy access to the separators yet to be processed (i.e. $\mathscr{S} \setminus \mathscr{T}$), along with printing answers when processed (in line 13, rather then when revealed in line 11), provides the polynomial delay. Correctness is derived directly by the correctness of the original algorithm, and the polynomial delay can be easily verified. In particular, the time between two consecutive results is $O(|\mathsf{V}(g)|^3)$.

4.3. Tractable expansion

Recall that Rose [40] proved that a chordal graph has fewer minimal separators than nodes. Combined with this result, Theorem 4.1 gives the first of the two conditions of Definition 2.

Corollary 4.3. *Let g be a graph. If I is a (maximal) independent set of $\mathcal{G}^{\text{ms}}(g)$, then $|I| < \mathsf{V}(g)$.*

Proof. Suppose that I is a maximal set of pairwise-parallel minimal separators of g. Then by Theorem 4.1, $h = g_{[I]}$ is a minimal triangulation of g, and *MinSep*(h) $= I$. The graph h is chordal, hence from Rose [40] we get that $|\mathit{MinSep}(h)| < |V(h)| = |V(g)|$. □

We now turn to proving the second condition of Definition 2. We do so by describing a general procedure for extending a set of pairwise-parallel minimal separators of a graph g to a maximal such set. Algorithm Extend of Fig. 3 can apply any known polynomial time triangulation heuristic, referred to as Triangulate, as a black box. It uses the following procedures as subroutines.

Algorithm Extend(g,φ)

1: $g_t := \mathsf{Triangulate}(g_{[\varphi]})$
2: $h := \mathsf{MinTriSandwich}(g_{[\varphi]}, g_t)$
3: **return** $\mathsf{ExtractMinSeps}(h)$

Fig. 3. An algorithm for extending a set φ of pairwise-parallel minimal separators.

- Saturate(g, S) receives a graph g and a set $S \subseteq V(g)$ of vertices, and saturates S (i.e., modifies g such that S becomes a clique).
- Triangulate(g) receives a graph g and returns a (not necessarily minimal) triangulation g' of g. We assume that this procedure runs in polynomial time. (For example, a naive implementation would be to add every possible edge; later we discuss smarter alternatives.)
- MinTriSandwich(g, g') receives a graph g and a triangulation g' of g, and returns a *minimal* triangulation of g. We note that, using one of the known algorithms [7,14,37], this procedure runs in time that is polynomial in the size of the graph.
- ExtractMinSeps(h) receives a chordal graph h and returns its set of minimal separators. Using the algorithm of Kumar [31], the execution time of this procedure is linear in h.

Extend takes as input a graph g and a set φ of pairwise-parallel minimal separators. It then proceeds by saturating the separators in φ, resulting in $g_{[\varphi]}$. At this stage it passes $g_{[\varphi]}$ to the triangulation heuristic Triangulate. We note that Triangulate does not have to produce a minimal triangulation. This is important since it allows us to incorporate *any* method for triangulation or tree decomposition. (We discuss in detail the translation between triangulations and tree decompositions in Section 5.)

The problem of transforming a non-minimal triangulation into a minimal one is called the *minimal triangulation sandwich problem* [24]. Various polynomial-time algorithms for this problem exist [14,37], and these were reported to perform well in practice [7].

So, at this stage we have a minimal triangulation h of $g_{[\varphi]}$. Theorem 4.4 (that we give in the next section) shows that h is also a minimal triangulation of g. Lemma 4.5 (also in the next section) shows that the set of minimal separators of h contains φ, which is essential as we need to *extend* φ. Finally, we can apply the algorithm of Kumar [31] to extract the minimal separators of the (chordal) graph h in linear time.

4.3.1. Correctness

To prove correctness of the algorithm Extend of Fig. 3, we need the following result by Heggernes [24].

Theorem 4.4 (*Heggernes [24]*)**.** *Given a graph g, let φ be an arbitrary set of pairwise-parallel minimal separators of g. Obtain a graph $g_{[\varphi]}$ by saturating each separator in φ.*

1. *$\varphi \subseteq$ ClqMinSep($g_{[\varphi]}$), that is, φ consists of clique minimal separators of $g_{[\varphi]}$.*
2. *ClqMinSep(g) $\subseteq$ MinSep($g_{[\varphi]}$); that is, every clique minimal separator of g is a (clique) minimal separator of $g_{[\varphi]}$.*
3. *Every minimal triangulation of $g_{[\varphi]}$ is a minimal triangulation of g.*

The next lemma builds on Theorems 4.1 and 4.4.

Lemma 4.5. *Let g be a graph, and φ a set of pairwise-parallel minimal separators of g. Let h be a minimal triangulation of $g_{[\varphi]}$. Then $\varphi \subseteq$ MinSep(h).*

Proof. By Part 1 of Theorem 4.4 we have that $\varphi \subseteq$ ClqMinSep($g_{[\varphi]}$). Since h is a minimal triangulation of $g_{[\varphi]}$ then by Part 2 of Theorem 4.1, h is the result of saturating a maximal set, say φ', of pairwise-parallel minimal separators of $g_{[\varphi]}$. Therefore, by Part 2 of Theorem 4.4 we have ClqMinSep($g_{[\varphi]}$) $\subseteq$ MinSep(h). This implies that $\varphi \subseteq$ MinSep(h), as claimed. □

We then conclude the correctness of the algorithm.

Lemma 4.6. *Let φ be a set of pairwise-parallel minimal separators of a graph g.* Extend(g, φ) *returns a maximal set I of pairwise-parallel minimal separators of g such that $\varphi \subseteq I$. Furthermore, the algorithm terminates in polynomial time.*

Proof. Assuming correctness of procedures Triangulate, and MinTriSandwich, the graph h is a minimal triangulation of $g_{[\varphi]}$. By Part 3 of Theorem 4.4, we have that h is a minimal triangulation of g. Consequently, from Part 2 of Theorem 4.1 we get that MinSep(h) $= I$ is a maximal set of pairwise-parallel minimal separators of g. By Lemma 4.5 it holds that $\varphi \subseteq$ MinSep(h), making I an extension of φ. All of the procedures in Fig. 3 run in time that is polynomial in the size of the graph making it polynomial as well. □

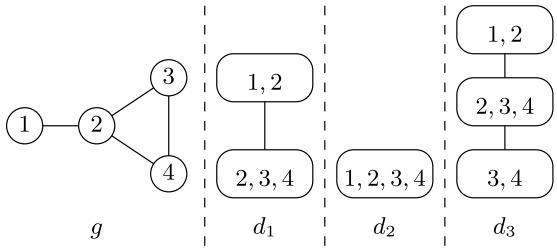


Fig. 4. A graph g and tree decompositions d_1, d_2 and d_3 of g. The decomposition d_1 is proper, but d_2 and d_3 are subsumed by d_1, and hence, improper.

From Corollary 4.3 and Lemma 4.6 we get the main result of this part.

Theorem 4.7. *The SGR* MSGraph *has a tractable expansion of independent sets.*

This theorem allows us to establish the main result of this paper.

4.4. Main result

From Theorems 3.1 and 4.7 we conclude that it is possible to enumerate the maximal independent sets of MSGraph in incremental polynomial time. Applying the bijection of Corollary 4.2, we get the main result of this paper.

Corollary 4.8. *Given a graph, the minimal triangulations can be enumerated in incremental polynomial time.*

In the next section, we will use this result for enumerating tree decompositions.

5. Enumerating the proper tree decompositions

In this section we define the notion of a *proper* tree decomposition, which is essentially a tree decomposition that is, intuitively, not deemed redundant due to another tree decomposition. Our ultimate goal is to enumerate *only* the proper tree decompositions, and we will show that this translates to enumerating the minimal triangulations.

5.1. Proper tree decompositions

Let d_1 and d_2 be two tree decompositions of a graph g. We say that d_1 and d_2 are *bag equivalent*, denoted $d_1 \equiv_{\mathrm{b}} d_2$, if $\mathrm{bags}(d_1) = \mathrm{bags}(d_2)$. We denote by $d_1 \sqsubseteq d_2$ the fact that for every bag $b_1 \in \mathrm{bags}(d_1)$ there exists a bag $b_2 \in \mathrm{bags}(d_2)$ such that $b_1 \subseteq b_2$.

Let g be a graph, and let d and d' be tree decompositions of g. We say that d' *strictly subsumes* d if $d' \sqsubseteq d$ and $\mathrm{bags}(d) \not\subseteq \mathrm{bags}(d')$ in multiset notation (i.e., some bag appears in d more times than it appears in d'). A tree decomposition is *proper* if it is not strictly subsumed by any tree decomposition, and it is *improper* otherwise.

Fig. 4 shows examples of proper and improper tree decompositions. It can be shown that d_1 is proper (e.g., since every clique of g is contained in some bag of d, as we prove in Proposition 5.3). But d_2 is not proper, since it is subsumed by d_1; that is, every bag of d_1 is contained in some bag of d_2, but the bag $\{1, 2, 3, 4\}$ is not a bag of d_1. For the same reason, d_2 is subsumed also by d_3. Finally, d_3 is subsumed by d_1 since every bag of d_1 is a bag of d_3, but the bag $\{3, 4\}$ is not a bag of d_1.

5.2. Enumeration

The main result of this section is the following, showing that enumerating the proper tree decompositions reduces to enumerating the minimal triangulations.

Theorem 5.1. *Let g be a graph. There is a bijection M between $\mathit{MinTri}(g)$ and the equivalence classes of $\equiv_{\mathrm{b}}$ over the proper tree decompositions of g. Moreover, given a minimal triangulation h of g, the proper tree decompositions in the class $M(h)$ can be enumerated with polynomial delay.*

Combined with Corollary 4.8, we get the following.

Corollary 5.2. *The set of proper tree decompositions of a given graph can be enumerated in incremental polynomial time.*

Next, we discuss the proof of Theorem 5.1, and in particular show how M is defined. We first need some propositions. The following proposition is a folklore, and it is using the fact that every collection of subtrees of a tree satisfies the *Helly property* [18].

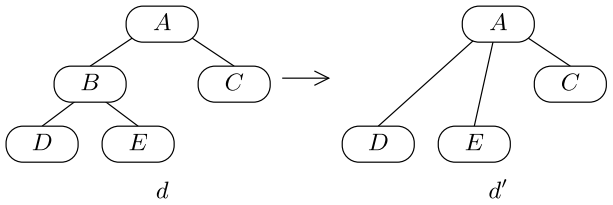


Fig. 5. Obtaining a strictly subsuming tree decomposition d' given a tree decomposition d with $B \subseteq C$.

Proposition 5.3. *If d is a tree decomposition of a graph g, then every clique of g is contained in some bag of d.*

Proof. We use the fact that the junction-tree property of a tree decomposition is equivalent to the property that for every node v of the graph, the bags of the tree decomposition that contain v form a (connected) subtree. Denote $d = (t, \beta)$ and let C be a clique of g. Every node v in C defines a subtree of t that is induced by the bags that contain v. Since d covers the edges of g, every two nodes in C must share some bag in d, and hence, their subtrees must share a vertex. It is known that every collection of subtrees of a tree satisfies the *Helly property* [18]: if every two subtrees share a vertex, then there exists a vertex that is shared by all the subtrees. In particular, there exists a vertex in d common to all of these subtrees; this shared node corresponds to a bag that contains C. □

The following proposition states that in a proper tree decomposition, there is no containment among bags.

Proposition 5.4. *If d is a proper tree decomposition of a graph g, then* bags(d) *is an antichain w.r.t. set inclusion (that is, no bag contains another).*

Proof. We need to show that a proper tree decomposition cannot have two bags with one contained in the other. Assume, by way of contradiction, that d is a proper tree decomposition of g with two bags $B, C \in \text{bags}(d)$ where $B \subseteq C$. Let A be the second bag in the path from B to C. Since d is a tree decomposition and A is on the path from B to C, we get that $B = B \cap C \subseteq A$.

Define d' to be the graph obtained from d by removing B and connecting A to all other neighbors of B, as illustrated in Fig. 5. We will show that d' is a tree decomposition for g. The first two properties of the tree decomposition still hold because A contains B. Consider the path between two bags α and β of d'. If the path between them is the same as in d, the third property still holds. If it changed, then the path used to go through B, and the only new bag that may appear in this path is A. In this case, $\alpha \cap \beta \subseteq B \subseteq A$, and the third property holds as well. We have found a tree decomposition d' for g that strictly subsumes d, hence d is improper, and this is a contradiction. □

From Theorem 2.3, the following easily follows.

Proposition 5.5. *If d is a tree decomposition of a graph g, then* saturate(g, d) *is a triangulation of g.*

Proof. According to the definitions, d is a tree decomposition of saturate(g, d). Hence, since every bag of d is a clique of saturate(g, d), it follows from Theorem 2.3 that saturate(g, d) is chordal. □

The definition of M is based on Lemma 5.6, stating that a chordal graph g has a single proper tree decomposition, up to the equivalence $\equiv_{\mathsf{b}}$, with the set of bags being precisely the set of maximal cliques.

Lemma 5.6. *If g is a chordal graph and d is a proper tree decomposition of g, then* bags(d) = *MaxClq*(g).

Proof. According to Proposition 5.3, every clique of g is contained in some bag of d, and according to Theorem 2.3, g has some tree decomposition, say d', where all the bags are cliques of g. So we have that $d' \sqsubseteq d$. If $\text{bags}(d) \not\subseteq \text{bags}(d')$, then d' strictly subsumes d, in contradiction to the fact that d is proper. Hence $\text{bags}(d) \subseteq \text{bags}(d')$, meaning that the bags of d are cliques of g. It thus follows that every maximal clique is a bag of d, or in notation, $\mathit{MaxClq}(g) \subseteq \text{bags}(d)$. Finally, Proposition 5.4 states that the bags of d are an antichain w.r.t. set inclusion, and hence, $\text{bags}(d) \subseteq \mathit{MaxClq}(g)$. We conclude that $\text{bags}(d) = \mathit{MaxClq}(g)$, as claimed. □

Based on Lemma 5.6, we define M to be the function that maps every $h \in \mathit{MinTri}(g)$ to the equivalence class of the proper tree decomposition of h. Lemma 5.7 states that M has the required properties.

Lemma 5.7. *Let g be a graph. The mapping M is a bijection between $\mathit{MinTri}(g)$ and the equivalence classes of $\equiv_{\mathsf{b}}$ over the proper tree decompositions of g.*

Proof. We show that M has the correct range, that it is surjective, and that it is bijective.

M **has a proper range**. Let h be a minimal triangulation of g, and let d be a proper tree decomposition of h in $M(h)$. Then d is also a tree decomposition of g, as the three properties of a tree decomposition still hold. We need to show that d is a *proper* tree decomposition of g. According to Lemma 5.6, we have $\text{bags}(d) = \mathit{MaxClq}(h)$, and therefore, $\mathsf{saturate}(g, d) = h$. Assume, by way of contradiction, that d is improper. Then d is strictly subsumed by some tree decomposition d' of g, meaning that $d' \sqsubseteq d$. Let h' be the graph $\mathsf{saturate}(g, d')$. From Proposition 5.5 it follows that h' is a triangulation of g. From $d' \sqsubseteq d$ and the fact that every bag of d is a clique of h, we conclude that $\mathsf{E}(h') \subseteq \mathsf{E}(h)$. And since h is a minimal triangulation, we get that $h = h'$. We can now conclude that d' is also a tree decomposition of g: the junction-tree property holds and the nodes are covered since it is a tree decomposition of g, and the edges are covered since those are the edges of h' that are covered by its definition. We get that both d and d' are tree decompositions of h, and d is strictly subsumed by d', which contradicts the fact that d is a proper tree decomposition of h.

M **is injective**. Let h_1 and h_2 be two minimal triangulations such that $h_1 \neq h_2$. Without loss of generality, assume that the edge $\{u, v\}$ is in h_1 but not in h_2. The nodes u and v are part of some maximal clique of h_1, so they share a bag in $M(h_1)$. But they are not part of any clique of h_2, so they do not share any bag in $M(h_2)$. Therefore, $M(h_1) \neq M(h_2)$.

M **is surjective**. Given a proper tree decomposition d of g, we need to show that there exists a minimal triangulation h of g such that $d \in M(h)$. Consider the graph $h = \mathsf{saturate}(g, d)$. We will show that h is a minimal triangulation, and that d belongs to $M(h)$.

We first show that h is a minimal triangulation of g. According to Proposition 5.5, h is a triangulation of g. Assume, by way of contradiction, that h is not minimal. Then there exists a minimal triangulation h' of g that is obtained from h by removing some edges; denote one of these edges by e. Consider a tree decomposition $d' \in M(h')$. The clique containing e in h is not a clique in h', and therefore $\text{bags}(d) \not\subseteq \text{bags}(d')$. Also note that since $h' \subseteq h$, every maximal clique of h' is contained in some maximal clique of h, and therefore $d' \sqsubseteq d$. Then d' strictly subsumes d, in contradiction to the fact that d is proper.

Finally, we need to show that d is a proper tree decomposition of h. The nodes of h are covered in d, and the junction-tree property holds, since d is a tree decomposition of g. The new edges of h are covered in d since they are all a result of saturation of the bags of d. So d is a tree decomposition of h, and we claim that it is proper. Assume, by way of contradiction, that d is not a proper tree decomposition of h, then the tree decomposition d' that strictly subsumes it is also a tree decomposition for g, contradicting the fact that d is a proper tree decomposition of g. □

To complete the proof of Theorem 5.1, we explain how the proper tree decompositions in the class $M(h)$ can be enumerated with polynomial delay for $h \in \mathit{MinTri}(g)$. Jordan [26] shows that, given a chordal graph h, a tree over the bags that represent the maximal cliques of h is a tree decomposition if and only if it is a maximal spanning tree, where the weight of an edge between two bags is the size of their intersection. Hence, this enumeration problem is reduced to enumerating all maximal spanning trees, which can be solved in polynomial delay [43]. Since Gavril [17] showed that in chordal graphs the number of maximal cliques of h is at most the number of nodes of h, we have a polynomial delay algorithm for enumerating the tree decompositions. This concludes the proof.

According to Corollary 5.2, we can enumerate all proper tree decompositions with incremental polynomial time. Note that this section also implies another alternative: we can enumerate only one representative of every equivalence class with the same complexity guarantees. That is, we can enumerate one proper tree-decomposition of each possible bag configuration with incremental polynomial time. The choice of which variation to use depends on the application at hand. For some applications, different tree-decompositions with the same bags may be of different quality, while for others only the bags matter.

6. Experimental evaluation

We now describe an experimental study over an implementation of our enumeration algorithm for minimal triangulations, namely the algorithm EnumMIS of Fig. 1 for the SGR $(\mathcal{G}^{\text{ms}}, A_{\mathsf{V}}^{\text{ms}}, A_{\mathsf{E}}^{\text{ms}})$, calling the procedure Extend of Fig. 3. The goal of the experimental study is twofold. First, we wish to understand how practical the execution cost of the algorithm is for enumerating many minimal triangulations (and tree decompositions). Second, we wish to study the ability of the algorithm to produce many *high-quality* triangulations, given an underlying triangulation algorithm (for Extend), and even to improve upon standard quality measures of the underlying algorithm itself. In Section 6.1 we describe the experimental setup, in Section 6.2 we report on the efficiency of the algorithm in terms of its delay, and in Sections 6.3 and 6.4 we study the quality of the results.

6.1. Experimental setup

We first describe the general setup for our study.

6.1.1. Implementation and hardware

We implemented all algorithms in C++, with STL data structures.[2] All experiments were carried out on a 2.6 GHz dual-core laptop with 8 GB of RAM running Windows 10 professional.

6.1.2. Triangulation algorithms

We implemented two well known triangulation algorithms as the procedure Triangulate in line 1 of the procedure Extend (Fig. 3). Both algorithms apply the general technique of *node-elimination ordering* [35], where nodes are eliminated from the graph in turn, by adding a subset of fill edges between the eliminated node and its neighbors in the (leftover) graph. Both algorithms guarantee a minimal triangulation (hence there was no need to call MinTriSandwich($g_{[\varphi]}, g_t$) in line 2 of Extend).

- MCS_M [4]. This is an extension of the *Maximum Cardinality Search* (MCS) algorithm for recognizing chordal graphs [41], which finds a minimal elimination ordering along with its corresponding minimal triangulation.
- LB_TRIANG [6]. This algorithms guarantees minimality of the triangulation by adding only a subset of the fill edges at each of the elimination steps, and allows for complete flexibility in determining the elimination order. We applied the *min fill* heuristic that selects, at each iteration, the node whose elimination results in the smallest number of edges to add.

6.1.3. Datasets

We used three types of datasets: probabilistic graphical models, database queries, and random (synthetic) graphs. For the first type, we used the following benchmark networks from the UAI probabilistic inference challenge.[3] The datasets Alchemy and DBN from the challenge are not described here as each of their graphs had only one or two minimal triangulations, and the enumeration ended instantaneously.

- **Promedas**: "PRObabilistic MEdical Diagnostic Advisory System". The Promedas Markov networks represent medical diagnosis cases, and consist of binary variables that were converted from layered noisy-or Bayesian networks. The dataset includes 33 graphs with 26-1039 nodes and 36-1696 edges, and many of them are considered too difficult for exact inference.[4]
- **Object detection**: Markov Random Fields for object-detection tasks in computer vision. It includes 79 instances of connected networks, each containing 60 nodes and between 135 to 180 edges.
- **Image segmentation**: Bayesian networks generated from image-segmentation tasks. It includes 6 graphs with 226-235 nodes and 617-647 edges.
- **Grids**: An $N \times N$ grid network. Such networks that are common in image processing [8], and networks that model problems such as medical diagnosis and object detection. This dataset includes 8 grids with $N = 10$ and $N = 20$, resulting in graphs with 100 or 400 nodes, and 180-760 edges.
- **Pedigree**: Bayesian networks used to model genetic information [16]. The data set includes 3 graphs, each has 385 nodes and 930 edges.
- **CSP**: Constraint-satisfaction problems. There are 3 instances in the dataset, with 67-100 nodes and 226-619 edges.

The datasets of second and third types are as follows.

- **TPC-H**: Graphs induced from TPC-H. These are the Gaifman (primal) graphs of joins for implementing the TPC-H benchmark queries in LogiQL, the Datalog variant of LogicBlox [1].[5] The queries include up to 22 nodes, and up to 46 edges, and their treewidth is up to 7.
- **Random**: Random $G(n, p)$ graphs in the Erdős–Rényi model. The number of nodes is n and every pair of nodes is connected by an edge with probability p (independently). We generated 54 random graphs for varying n between 30 and 200, and three values of p: 0.3 (sparsest), 0.5 and 0.7 (densest).

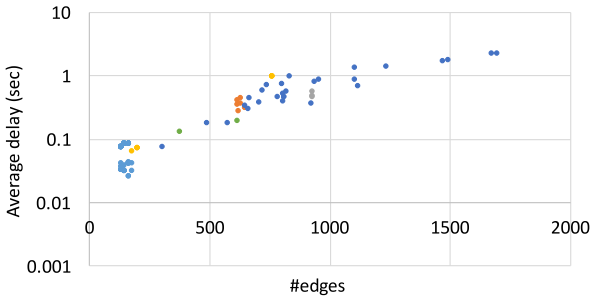
As a baseline approach, we implemented the algorithm of DunceCap [42] for generating all of the generalized hypertree decompositions (each involving an underlying tree decomposition). However, this algorithm is designed to handle small join queries and to span a much greater space of objects (namely, the generalized hypertree decompositions). In particular, on the TPC-H dataset we observed that on the smaller queries our algorithm is faster by 3 to 4 orders of magnitude, and on some of the larger queries (Q7 and Q9) we could not get their algorithm to terminate in less than two hours (while our algorithms terminated in a few seconds, as we later discuss). Therefore, we decided to exclude comparisons to this implementation. As of today, we are not aware of any other published algorithms for enumerating (minimal) triangulations or tree decompositions with guarantees of correctness (completeness).

2 The code is available online: https://github.com/NofarCarmeli/MinTriangulationsEnumeration.

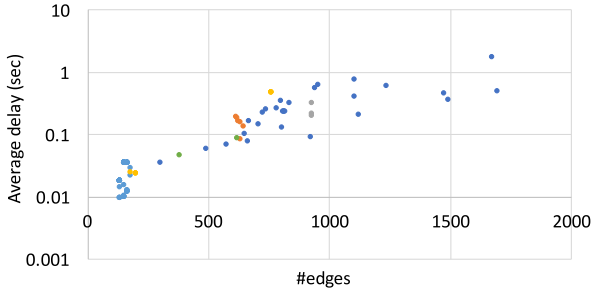
3 http://www.cs.huji.ac.il/project/PASCAL/showNet.php.

4 http://graphmod.ics.uci.edu/uai08/Evaluation/Report/Benchmarks.

5 The queries, provided to us by LogicBlox, are used for benchmarking the engine.



(a) LB_TRIANG



(b) MCS_M

Fig. 6. Average delay (in seconds) for the two triangulation algorithms over the probabilistic-graphical-model benchmarks: Object Detection (●), Segmentation (●), Pedigree (●), Grids (●), Promedas (●), CSP (●).

6.2. Execution cost

In what follows we report on the delay of the two variants of the implementation, corresponding to the two triangulation algorithms LB_TRIANG and MCS_M.

6.2.1. Probabilistic graphical models

We measured the average delay between minimal triangulation printouts for the network datasets from the UAI challenge. The measurements were conducted during 30 minutes executions. 5 of the graphs in Promedas, and one graph of CSP completed the enumeration within this time. We plotted the delay of the other graphs against the number of their edges. The plots, corresponding to the two minimal triangulation algorithms LB_TRIANG and MCS_M, are presentedin Figs. 6a and 6b, respectively, using log-scale. Overall, we see that the delay increases with the size of the graph. However, this trend varies between the different benchmarks. While this dependency is apparent for the Promedas data set, the average delay for object detection has little correlation with the number of edges in the graph.

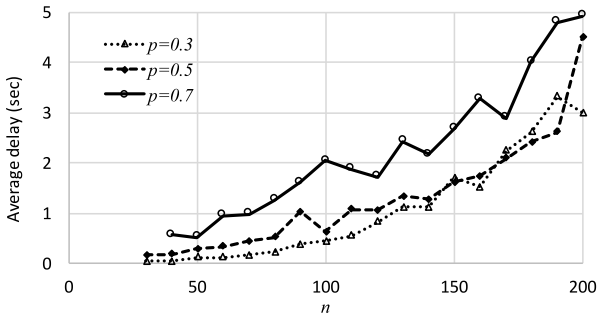
6.2.2. Random graphs

We measured the average delay (in seconds) between the printout of consecutive minimal triangulations during a 30 min execution. The plots in Figs. 7a and 7b show the average delay vs. the size of the graph for the two variants. We can see that the delay increases with the size of the graph, and that the general trend is that the delay is larger for denser graphs. We also see that for LB_TRIANG the delay is generally longer than for MCS_M.

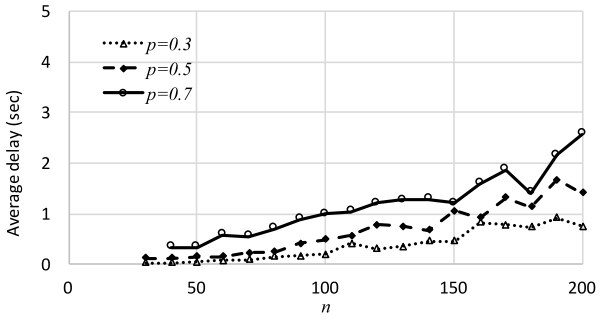
6.2.3. Database queries

We evaluated our enumeration algorithm over a set of 22 queries from the TPC-H dataset. The graphs of these queries are quite small when compared to the UAI datasets (< 23 nodes). Moreover, half of these graphs are chordal to begin with (i.e., have only one minimal triangulation—the graph itself), and hence, irrelevant for us. Except for two queries, all of the rest had at most 5 minimal triangulations. The remaining two queries are Q7 (Volume shipping Query) and Q9 (Product Type Profit Measure Query), and they have a considerable number of minimal triangulations: 700 and 588, respectively. When considering the minimum-width tree decomposition for each of the queries, the largest bag was of size 8; this is due to a relation of arity 8 in the query. In fact the largest bag in each of the queries had at most two variables more than the size of the largest relation. The execution for all 22 queries completed within 5 s.

In one of the queries we compared the delays for two modes of printing: the one of EnumMIS and the one of EnumMISHold that prints upon extraction from the queue, as described in Section 3.2.2. We refer to the former as UG



(a) LB_TRIANG



(b) MCS_M

Fig. 7. Average delay over 54 graphs randomly generated from the Erdős–Rényi $G(n, p)$ for varying n and p.

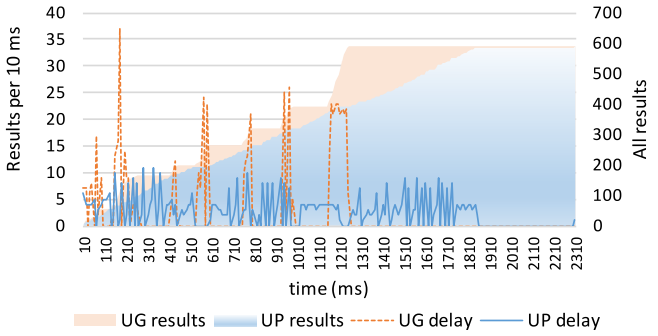


Fig. 8. Delay behavior in two printing modes: UG (Upon Generation, as in EnumMIS), and UP (Upon Pop, as in EnumMISHold).

(Upon Generation) and to the latter as UP (Upon Pop). Recall that both modes guarantee incremental polynomial time (Theorem 3.4). This gives us a sense of the practical impact of printing the solutions as soon as possible compared to holding the solutions to attain an easy-to-prove incremental polynomial time algorithm. The results are in Fig. 8. While the dotted line (of UG) has bursts of high-frequency prints followed by periods where no new triangulation is created, the solid line (UP) has a more steady pace as can be seen by the constant slope in Fig. 8. As expected, despite the fact that the last result of UG is printed earlier that of UP, termination is at the same time in both modes, as the algorithm still needs to check that there are no additional minimal triangulations.

6.3. Quality

In what follows we report on the quality of the generated minimal triangulations in terms of two standard measures of quality for triangulations and tree decompositions: *fill* and *width*. *Fill* refers to the total number of edges added in order to make the resulting graph chordal, while *width* refers to the size of the largest clique in the generated triangulation (minus one).[6] The natural benchmark for the quality of the triangulations is the first result our enumeration returns, as it is the result we would get by running the minimal triangulation algorithm we used, on the original input graph.

[6] Recall that is NP-hard to find a triangulation that minimizes the fill [44] or the width [2].

Table 1
Width statistics on generated triangulations following 30 min execution.

Dataset	#trng	min-w	#≤$\mathbf{w}_1$ (%)	%$\mathbf{w}\downarrow$ (max)
MCS_M				
Promedas (28)	11064.5	25.8	3713.6 (33.6%)	2.2 (15.2)
Grids (8)	40319.8	18.4	93.6 (0.2%)	0.0 (0.0)
Obj. Detection (79)	100349.9	6.1	42743.9 (42.6%)	0.4 (12.5)
Segmentation (5)	12836.5	23.0	20.5 (0.2%)	0.0 (0.0)
Pedigree (3)	7789.0	31.7	3087.3 (39.6%)	0.0 (0.0)
CSP (2)	29450.5	16.5	26741.5 (90.8%)	13.2 (26.3)
LB_TRIANG				
Promedas (28)	4220.7	18.6	2352.0 (55.7%)	1.9 (16.7)
Grids (8)	13881.3	24.5	1273.0 (9.2%)	3.0 (8.7)
Obj. Detection (79)	33295.4	5.8	15709.3 (47.2%)	0.0 (0.0)
Segmentation (5)	5174.2	21.8	2141.8 (41.4%)	10.3 (20.7)
Pedigree (3)	3646.0	23.7	3227.7 (88.5%)	5.3 (14.8)
CSP (2)	11772.0	16.5	3760.5 (31.9%)	0.0 (0.0)

Table 2
Fill statistics on generated triangulations following 30 min execution.

Dataset	#trng	min-f	#≤$\mathbf{f}_1$ (%)	%$\mathbf{f}\downarrow$ (max)
MCS_M				
Promedas (28)	11064.5	3353.4	8136.0 (73.5%)	18.1 (49.9)
Grids (8)	40319.8	2752.6	15771.4 (39.1%)	4.2 (28.1)
Obj. Detection (79)	100349.9	30.0	27614.1 (27.5%)	19.9 (47.1)
Segmentation (5)	12836.5	2555.2	5269.7 (41.1%)	5.9 (12.5)
Pedigree (3)	7789.0	3525.7	743.0 (9.5%)	2.8 (3.5)
CSP (2)	29450.5	46.0	18815.5 (63.9%)	35.2 (55.8)
LB_TRIANG				
Promedas (28)	4220.7	1239.4	175.0 (4.1%)	0.2 (11.1)
Grids (8)	13881.3	1600.3	1.0 (0.0%)	0.0 (0.0)
Obj. Detection (79)	33295.4	27.6	5110.7 (15.3%)	10.4 (27.6)
Segmentation (5)	5174.2	1402.0	130.2 (2.5%)	1.2 (4.2)
Pedigree (3)	3646.0	1491.0	1.0 (0.0%)	0.0 (0.0)
CSP (2)	11772.0	34.5	664.0 (5.6%)	1.4 (3.0)

For each graph of the probabilistic inference dataset, we executed the enumeration algorithm for 30 min. The results in Table 1 include only the experiments where the enumeration did not complete. For each graph we measured the following: the number of generated triangulations (**#trng**), the minimum observed width over all printed triangulations (**min-w**), the number of printed triangulations of width at most that of the first (#≤$\mathbf{w}_1$), the average reduction in width (over the dataset) and the maximum improvement in parentheses ($\mathbf{w}\downarrow$ (%)). In Table 2 we show the same results for fill instead of width (**min-f**, #≤$\mathbf{f}_1$ and $\mathbf{f}\downarrow$ (%)).

We can see that the algorithm, in both variants, is able to generate a significant number of triangulations of high quality, in terms of both width and fill. Moreover, it amplifies the quality of the underlying triangulation, by means of width, and much more by means of fill. According to the number of triangulations printed, MCS_M enables generating twice as many triangulations as LB_TRIANG. However, with the exception of only a handful of the graphs tested, the triangulations generated by LB_TRIANG are superior in both the width and fill metrics (this is especially apparent for the Promedas and Pedigree datasets). Furthermore, this set of superior triangulations accounts for a larger portion of the total number of triangulations that generated.

6.4. Case study

In this section we take a closer look at the behavior of the enumeration during a single execution. We use a graph from the Promedas dataset. In Fig. 9 we show the cumulative number of results generated over time. We consider three types of results: *(a)* all minimal triangulations, *(b)* minimal triangulations of the minimum width (where the minimum is taken over the printed triangulations), *and (c)* those with a width at most that of the first triangulation (≤w1), which is the one that we would obtain by using only the triangulation algorithm at hand. The reduction in the number of new triangulations over time is consistent with the increase in the delay entailed in the guarantee of incremental polynomial time, rather than polynomial delay.

Fig. 10 presents the reduction in the minimum width and minimum fill obtained during the execution of the algorithm. Each time slice records the minimum width (solid curve) and minimum fill (dotted curve) observed up to that time. We can see that both the width and the fill reduce over time, but the minimum observed width is reached very quickly, while attaining the minimum observed fill takes longer.

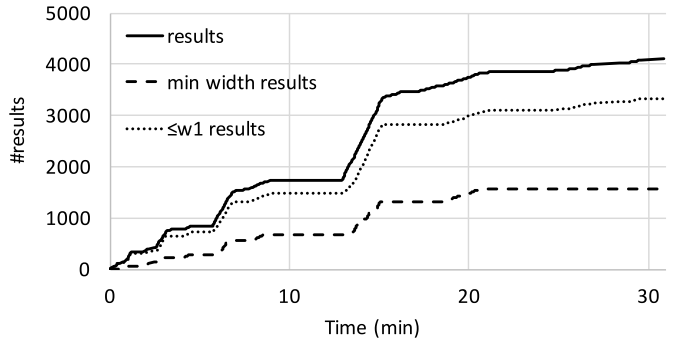


Fig. 9. Cumulative number of triangulations.

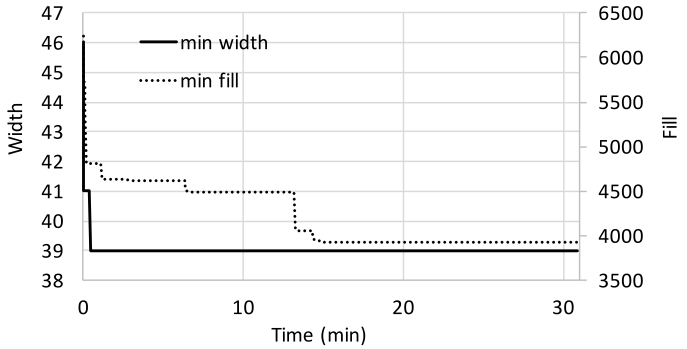


Fig. 10. Minimum width and fill over time.

7. Concluding remarks

We introduced the concept of a succinct graph representation (SGR), and presented an enumeration algorithm for its maximal independent sets. The algorithm enumerates in incremental polynomial time under complexity assumptions: the SGR is tractably accessible, and it has a tractable expansion. Consequently, we established an algorithm for enumerating the minimal triangulations of a graph by reducing the problem to the enumeration of the maximal independent sets of an SGR, and showing that the complexity assumptions hold. We also proved that enumerating the minimal triangulations enables the enumeration of the proper tree decompositions. Our experimental study showed that the algorithm is effective on graphs of various domains, and is able to enhance off-the-shelf algorithms for triangulation (or tree decomposition) by generating many (rather than just one) high-quality different triangulations, and even improve standard quality measures such as width and fill.

This work opens up quite a few directions for future work. On the theoretical side, it is left open whether the enumeration of the minimal triangulations can be carried out with polynomial delay. As discussed in Sections 2.6 and 3.3, it is not clear how to do this with known techniques, and the abstraction used here cannot achieve this time bound. Polynomial delay is possible in the case that the number of minimal separators of the input graph is polynomial in the size of the input graph. In a follow up work to the original publication of this manuscript, Ravid et al. [38] showed how to perform in such cases ranked enumeration under a wide class of cost functions that generalizes width and fill-in. If the number of minimal separations is not bounded, the question of incorporating some order remains open. In terms of the space complexity, it is left open to determine whether the problem can be solved using polynomial (or even sub-exponential) space; the algorithm presented here uses exponential space in the worst case as it stores the minimal separators of the graph and the generated results. From the practical aspect, the algorithm presented here holds many opportunities for optimization over real-life graphs. An optimized version of the code is available online.[7]

Acknowledgments

This work was supported in part by the US–Israel Binational Science Foundation (BSF) Grant No. 2014391, the Israel Science Foundation (ISF) Grant No. 1295/15, and the Austrian Science Fund (FWF): P30930-N35, W1255-N23. Benny

[7] https://github.com/TechnionTDK/efficient-td-enum.

Kimelfeld is a Taub Fellow, supported by the The Henry and Marilyn Taub Foundation, Israel. The authors are greatly thankful to LogicBlox, and in particular Hung Ngo, for insightful discussions and for providing test data.

References

[1] M. Aref, B. ten Cate, T.J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T.L. Veldhuizen, G. Washburn, Design and implementation of the logicblox system, in: SIGMOD, ACM, 2015, pp. 1371–1382.
[2] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in ak-tree, SIAM J. Algebr. Discrete Methods 8 (2) (1987) 277–284.
[3] D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Appl. Math. 65 (1–3) (1996) 21–46.
[4] A. Berry, J.R.S. Blair, P. Heggernes, Maximum cardinality search for computing minimal triangulations, in: WG, in: WG '02, Springer-Verlag, London, UK, UK, 2002, pp. 1–12.
[5] A. Berry, J.P. Bordat, O. Cogis, Generating all the minimal separators of a graph., in: P. Widmayer, G. Neyer, S. Eidenbenz (Eds.), WG, in: Lecture Notes in Computer Science, vol. 1665, Springer, 1999, pp. 167–172.
[6] A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet, Y. Villanger, A wide-range algorithm for minimal triangulation from an arbitrary ordering, J. Algorithms 58 (1) (2006) 33–66.
[7] J.R. Blair, P. Heggernes, J.A. Telle, A practical algorithm for making filled graphs minimal, Theoret. Comput. Sci. 250 (1–2) (2001) 125–141.
[8] A. Blake, P. Kohli, C. Rother, Markov Random Fields for Vision and Image Processing, The MIT Press, 2011.
[9] E. Boros, K.M. Elbassioni, V. Gurvich, Algorithms for generating minimal blockers of perfect matchings in bipartite graphs and related problems, in: ESA, in: Lecture Notes in Computer Science, vol. 3221, Springer, 2004, pp. 122–133.
[10] S. Cohen, I. Fadida, Y. Kanza, B. Kimelfeld, Y. Sagiv, Full disjunctions: Polynomial-delay iterators in action, in: VLDB, ACM, 2006, pp. 739–750.
[11] S. Cohen, B. Kimelfeld, Y. Sagiv, Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties, J. Comput. System Sci. 74 (7) (2008) 1147–1159.
[12] A. Conte, R. Grossi, A. Marino, T. Uno, L. Versari, Listing maximal independent sets with minimal space and bounded delay, in: G. Fici, M. Sciortino, R. Venturini (Eds.), String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings, in: Lecture Notes in Computer Science, 10508, Springer, 2017, pp. 144–160.
[13] A. Conte, T. Uno, New polynomial delay bounds for maximal subgraph enumeration by proximity search, in: M. Charikar, E. Cohen (Eds.), Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019, ACM, 2019, pp. 1179–1190.
[14] D. Dahlhaus, Chapter minimal elimination ordering inside a given chordal graph, in: WG, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 132–143.
[15] G.A. Dirac, On rigid circuit graphs, in: Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg, Vol. 25, Universität Hamburg, 1961.
[16] M. Fishelson, D. Geiger, Exact genetic linkage computations for general pedigrees, in: ISMB/ECCB, 2002, pp. 189–198.
[17] F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, J. Combin. Theory 16 (1974) 47–56.
[18] M.C. Golumbic, Chapter 4 - triangulated graphs, in: M.C. Golumbic (Ed.), Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980, pp. 81 – 104.
[19] M. Golumbic, H. Kaplan, R. Shamir, Graph sandwich problems, J. Algorithms 19 (3) (1995) 449–473.
[20] G. Gottlob, G. Greco, F. Scarcello, Pure Nash equilibria: Hard and easy games, J. Artif. Intell. Res. (JAIR) 24 (2005) 357–406.
[21] G. Gottlob, M. Grohe, N. Musliu, M. Samer, F. Scarcello, Hypertree decompositions: Structure, algorithms, and applications, in: WG, in: Lecture Notes in Computer Science, vol. 3787, Springer, 2005, pp. 1–15.
[22] G. Gottlob, N. Leone, F. Scarcello, Hypertree decompositions and tractable queries, J. Comput. System Sci. 64 (3) (2002) 579–627.
[23] G. Gottlob, Z. Miklós, T. Schwentick, Generalized hypertree decompositions: NP-hardness and tractable variants, J. ACM 56 (6) (2009).
[24] P. Heggernes, Minimal triangulations of graphs: A survey, Discrete Math. 306 (3) (2006) 297–317, Minimal Separation and Minimal Triangulation.
[25] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis, On generating all maximal independent sets, Inf. Process. Lett. 27 (3) (1988) 119–123.
[26] M. Jordan, An Introduction to Probabilistic Graphical Models, University of California, Berkeley, 2002, (chapter 17).
[27] O. Kalinsky, Y. Etsion, B. Kimelfeld, Flexible caching in trie joins, 2016, CoRR abs/1602.08721.
[28] B. Kenig, A. Gal, On the impact of junction-tree topology on weighted model counting, in: SUM, in: Lecture Notes in Computer Science, vol. 9310, Springer, 2015, pp. 83–98.
[29] T. Kloks, D. Kratsch, J.P. Spinrad, On treewidth and minimum fill-in of asteroidal triple-free graphs, Theoret. Comput. Sci. 175 (2) (1997) 309–335.
[30] P.G. Kolaitis, M.Y. Vardi, Conjunctive-query containment and constraint satisfaction, J. Comput. System Sci. 61 (2) (2000) 302–332.
[31] P. Kumar, C. Madhavan, Minimal vertex separators of chordal graphs, Discrete Appl. Math. 89 (1–3) (1998) 155–168.
[32] S. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, J. Royal Statist. Soc. B, 50 (2) (1988) 157–224.
[33] E.L. Lawler, J.K. Lenstra, A.H.G.R. Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, SIAM J. Comput. 9 (3) (1980) 558–565.
[34] D. Marx, Approximating fractional hypertree width, ACM Trans. Algorithms 6 (2) (2010).
[35] T. Ohtsuki, L.K. Cheung, T. Fujisawa, Minimal triangulation of a graph and optimal pivoting order in a sparse matrix, J. Math. Anal. Appl. 54 (3) (1976) 622–633.
[36] A. Parra, P. Scheffler, Characterizations and algorithmic applications of chordal graph embeddings, Discrete Appl. Math. 79 (1–3) (1997) 171–188.
[37] B.W. Peyton, Minimal orderings revisited, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 271–294.
[38] N. Ravid, D. Medini, B. Kimelfeld, Ranked enumeration of minimal triangulations, in: Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, in: PODS '19, ACM, New York, NY, USA, 2019, pp. 74–88.
[39] N. Robertson, P. Seymour, Graph minors. III. Planar tree-width, J. Combin. Theory Ser. B 36 (1) (1984) 49–64.
[40] D.J. Rose, Triangulated graphs and the elimination process, J. Math. Anal. Appl. 32 (3) (1970) 597–609.
[41] R.E. Tarjan, M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM J. Comput. 13 (3) (1984) 566–579.
[42] S. Tu, C. Ré, DunceCap: Query plans using generalized hypertree decompositions, in: SIGMOD, ACM, 2015, pp. 2077–2078.
[43] T. Yamada, S. Kataoka, K. Watanabe, Listing all the minimum spanning trees in an undirected graph, Int. J. Comput. Math. 87 (14) (2010) 3175–3185.
[44] M. Yannakakis, Computing the minimum fill-in is NP-complete, SIAM J. Algebr. Discrete Methods 2 (1) (1981) 77–79.
[45] J. Zhao, R.L. Malmberg, L. Cai, Rapid *ab initio* RNA folding including pseudoknots via graph tree decomposition, in: WABI, 2006, pp. 262–273.