Counting Solutions Under Cardinality Constraints: Structure Counts in Counting

Max Bannach, Markus Hecher

Counting Solutions under Cardinality Constraints: Structure Counts in Counting

Max Bannach[1] , **Markus Hecher**[2]

[1]European Space Agency, AI and Data Science Section, Noordwijk, The Netherlands
[2]University of Artois, CNRS, UMR8188, Computer Science Research Center of Lens (CRIL), France
max.bannach@esa.int, hecher@mit.edu

Abstract

Model counting is a powerful extension of constraint reasoning that, instead of finding *a* solution to a constraint system, allows to identify the *number* of such solutions. Cardinality constraints are used to filter solutions of a certain *quality* by restricting the number of elements that can be added to the solution. Naturally, one would like to combine both in order to count the number of solutions of good quality. Unfortunately, the two concepts do not get along so well as (1) cardinality constraints may not be parsimonious (due to auxiliary variables, the system's number of solutions may change in an uncontrolled way) and (2) such constraints may destroy structural properties, which are crucial for the performance of modern solvers. This article provides a systematic study of existing cardinality constraints in the light of model counting, observing that none of them are both, parsimonious and treewidth-preserving. We present structure-aware cardinality constraints that are parsimonious *and* guaranteed to increase the input's treewidth only in a controlled way. Detailed experiments reveal that our encodings outperform existing ones.

1 Introduction

One of the most important techniques to encode constraint satisfaction problems and problems from other domains to SAT are *cardinality constraints.* A cardinality constraint over literals $\ell_1, \dots, \ell_p$ has the form

$$\sum_{i=1}^{p} \ell_i \bowtie k \quad \text{for} \quad \bowtie \in \{<, \leq, =, \geq, >\}$$

for a target value $k \in \mathbb{N}$. Such constraints arise naturally if optimization problems are tackled via SAT technology. For a running example, let us consider the *odd cycle transversal* problem (OCT), which asks whether we can delete in a given graph G at most k vertices in order to make it *bipartite* (i.e., such that it does not contain an odd cycle, hence the name). This problem can be considered as a variation of the three coloring problem in which one color is a "joker color" that is allowed to be adjacent to itself, but which can be given to at most k vertices (these are the ones to be deleted). Figure 1 contains an example graph with an highlighted solutions and a typical SAT formula with a cardinality constraint.

We can choose from a wide variety of possible encodings to realize cardinality constraints (Roussel and Manquinho 2021, Chapter 2). The perhaps simplest encoding is the *naive encoding,* which expresses that for every subset S of $\ell_1, \dots, \ell_p$ of size $k+1$, at least one literal in S must be false:

$$\sum_{i=1}^{p} \ell_i \leq k \quad \equiv \quad \bigwedge_{\substack{S \subseteq \{\ell_1, \dots, \ell_p\} \\ |S| = k+1}} \bigvee_{\ell \in S} \neg \ell.$$

This and many other encodings have been intensively studied in the literature, concerning their size (in terms of clauses and introduced auxiliary variables), as well as their propagation properties and empirical performance – see the survey by Wynn (2018) for an overview.

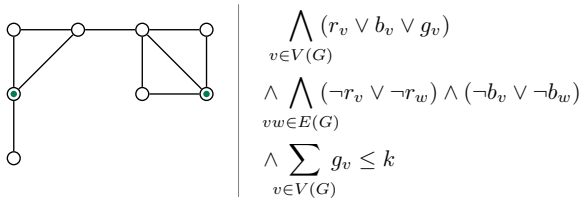


Figure 1: An example graph with an odd cycle transversal of size two (left), containing 6 OCTs of size two. On the right a typical SAT encoding φ_{oct} of the OCT problem using a cardinality constraint.

An advantage of using SAT technology with cardinality constraints instead of, say, integer linear programs is that SAT naturally generalizes to *counting problems* and that optimized *model counters* are available (Fichte, Hecher, and Hamiti 2021; Fichte et al. 2023; Korhonen and Järvisalo 2023). Assume instead of searching *one* size-k odd cycle transversal of a given graph, we wish to know *how many* such OCTs there are (see Figure 1). The counting version of SAT is called #SAT and asks, give a propositional formula φ, to compute the number of models $\#(\varphi)$ of φ. Unfortunately, we can*not* simply take the formula φ_{oct} from Figure 1 and hand it to a #SAT solver in order to obtain the number of OCTs of G, since we have no control of the behavior of a CNF encoding of $\sum_{v \in V(G)} g_v \leq k$. The cardinality constraint may introduce various auxiliary variables and potentially will increase the number of models of the formula beyond the number of size-k OCTs. Our first question, thus, is which encodings preserve the solution count?

Q1: Which cardinality encodings are parsimonious?

The next natural consideration is to identify under the parsimonious encodings those that empirically perform well

in the model counting scenario. This line of reasoning quickly leads to another question: While propagation properties are essential for answering satisfiability queries, they are less relevant in the counting setting as all the models need to be considered anyway. Fundamentally more important for counting are *structural properties* such as the input's treewidth (formal definition of such properties will be given in the following sections). Indeed, it is assumed that efficient model counting is only possible on well-structured instances, which is reflected by the fact that all state-of-the-art counters utilize structure of the input in one way or the other (Fichte, Hecher, and Hamiti 2021; Korhonen and Järvisalo 2023). Let $\mathrm{tw}(\varphi)$ be the treewidth of the *primal graph* of φ (again, formal definitions will follow later; the primal graph contains a vertex for every variable and connects variables appearing together in clauses). By the quoted works, $\#(\varphi)$ can only be computed efficiently if $\mathrm{tw}(\varphi)$ is small. Hence, a crucial property for an encoding of a cardinality constraint $\sum_{i=1}^{p} \ell_i \leq k$ is not necessarily how many auxiliary variables it requires, but rather how $\mathrm{tw}\big(\varphi \wedge (\sum_{i=1}^{p} \ell_i \leq k)\big)$ compares to $\mathrm{tw}(\varphi)$. Let us say that an encoding is *structure-aware* (SAW) if there is a function $f\colon \mathbb{N} \to \mathbb{N}$ such that $\mathrm{tw}\left(\varphi \wedge (\sum_{i=1}^{p} \ell_i \leq k)\right) \leq f(\mathrm{tw}(\varphi))$.

Q2: Which cardinality encodings are structure-aware?

However, note that we can*not* hope to obtain a cardinality constraint for an arbitrarily slowly growing function f. The reason is the following fact[1]:

Fact 1 (folklore)**.** *Any circuit with n inputs of size $\mathrm{poly}(n)$ that decides whether at least half of its inputs are true requires a treewidth of $\Omega(\log n)$.*

It is also easy to see that for any CNF φ of incidence treewidth w (the incidence treewidth is the treewidth of the bipartite graph that contains a vertex for every variable and every clause, and that connects variables to the clauses containing them) there is an equivalent circuit C of treewidth w (the circuit is essentially the incidence graph with an or-gate for every clause and a single big-and gate connected to all the or-gates). Since the incidence treewidth of a formula is a lower bound for $\mathrm{tw}(\varphi)$ (Samer and Szeider 2021), we get:

Corollary 1. *Any* CNF *φ encoding a constraint $\sum_{i=1}^{n} x_i \bowtie k$ has $\mathrm{tw}(\varphi) \in \Omega(\log k)$.*

1.1 Contribution of this Article

The contributions of this article are four fold: First, we provide an overview of existing encodings and analyze their impact on the input's treewidth. We also empirically analyze the well-established implementation of many of these encodings available in the PySAT library (Ignatiev, Morgado, and Marques-Silva 2018) to see which of them preserve the number of solutions (but note that different implementations of the same encoding could lead to different outcomes).

Second, we provide four cardinality constraints that are both, *parsimonious* and *structure-aware*. Two of them are build on a sequential or binary counter, respectively, which we evaluate distributed along a tree decomposition. Two more are obtained by balancing the given tree decomposition before applying the structure-aware counters, which coincidentally leads to structure-aware *totalizer encodings*. We, third, provide implementations of our structure-aware cardinality constraints in a first-order language. As a positive side effect, our tool chain seemingly generalizes to cardinality constraints in *answer set programming* (ASP).

Finally, we created and make available to the scientific community a benchmark set for #SAT based on highly structured formulas that additionally contain cardinality constraints. The benchmark set is described in Section 6 and available on zenodo DOI 10.5281/zenodo.16884175.

Preliminary experiments reveal that these formulas are rather challenging for modern model counters if cardinality constraints from PySAT are used – supporting our hypothesis that constraints that destroy the inputs structure are bad for model counting. In contrast, we observe significantly better performance on formulas in which the cardinality constraint was realized with structure-aware encodings.

1.2 Related Work

Cardinality constraints belong to the most important constraints in conceptional modeling (Olivé 2007, Chapter 4), knowledge representation (Berre et al. 2018), constraint programming (Rossi, van Beek, and Walsh 2006, Chapter 7), and artificial intelligence (Russell and Norvig 2020, Chapter 5). They are therefore an active field of research (Jo and Cho 2025; Liang and Lin 2024; Zhang, Chen, and Hu 2023; Leung and Wang 2022). For surveys, see for instance Wynn (2018) or Tillmann et al. (2024).

Model counting has a long history in computational complexity theory (Papadimitriou 2007, Chapter 18), and is a central tool in symbolic artificial intelligence (Shaw and Meel 2024) and probabilistic reasoning (Chavira and Darwiche 2008). Cardinality constraints, however, received less attention in this context. They were studied mainly for counting in first-order logic theories (Tóth and Kuzelka 2024; Malhotra and Serafini 2022) and to count the solution of global constraints (Bianco et al. 2019).

It is folklore that model counting currently is only tractable on structured instances, see for instance the discussions in Korhonen and Järvisalo (2023) and Fichte, Hecher, and Hamiti (2021). We are, however, not aware of a work that considers structural properties of cardinality constraints. The closest is by Pichler et al. (2010) who study weighted constraints in answer set programming, yet this does not directly relate to encoding techniques for such constraints.

1.3 Structure of this Article

Below, we provide the necessary background. In Section 3, we review existing encodings of cardinality constraints and theoretically analyze whether they are parsimonious or structure-aware. We propose novel encodings in Section 4 that are both parsimonious *and* structure-aware. Finally, we provide an implementation of these encodings in Section 5 and empirically analyze them in Section 6.

[1] See, e.g., the discussion in cstheory.stackexchange.com/questions/26021/minimum-tree-width-of-circuit-for-majority.

2 Preliminaries

An *assignment* of a propositional formula φ is a mapping $\beta\colon \operatorname{vars}(\varphi) \to \{0,1\}$ from its variables to the truth values. If φ evaluates to true under β, we write $\beta \models \varphi$ and say that β is a *model* of φ. We define $\operatorname{models}(\varphi) := \{\, \beta \mid \beta \models \varphi \,\}$ and let $\#(\varphi) = |\operatorname{models}(\varphi)|$. The problem of computing the value $\#(\varphi)$ is called the *model counting problem* or #SAT.

In this article, we assume formulas φ to be in *conjunctive normal form* (CNF), i.e., to be a conjunction of disjunctions of literals. Unless stated otherwise, we denote the variables of φ with $x_1, \ldots, x_n$ and its clauses with $c_1, \ldots, c_m$. Formulas in that form can be represented as an undirected graph G_φ with vertices $V(G_\varphi) = \{x_1, \ldots, x_n\}$ and edges $\{\, \{x_i, x_j\} \mid x_i \text{ and } x_j \text{ appear together in some clause } c \,\}$. This graph is known as the *primal graph* of the formula and can be used to describe *structural properties* of the formula.

2.1 Structure in Graphs and Formulas

A *tree decomposition* (TD) of a graph G is a pair (T, χ) in which T is a rooted tree and $\chi\colon V(T) \to 2^{V(G)}$ a mapping from the nodes of T to sets of vertices of G, called *bags*. A tree decomposition must satisfy the following properties:

Connectedness For every $v \in V(G)$ the induced subgraph $T[\{t \mid v \in \chi(t)\}]$ is non-empty and connected.

Covering For every edge $uv \in E(G)$ there is a bag t with $\{u, v\} \subseteq \chi(t)$.

The *width* of a tree decomposition is $\max_{t \in V(T)} |\chi(t)| - 1$, and the *treewidth* $\operatorname{tw}(G)$ of G is the minimum width any tree decomposition of G must have. Figure 2 provides an example of a tree decomposition. We define the treewidth of a formula as $\operatorname{tw}(\varphi) := \operatorname{tw}(G_\varphi)$.

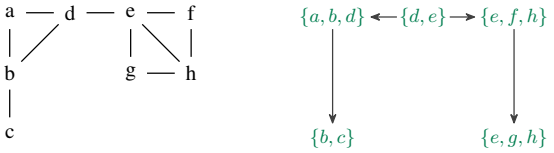


Figure 2: The treewidth of the graph from the introduction (left) is at least two since it contains a cycle. It is also at most two by the tree decomposition on the right.

It will often be convenient in the description of algorithms to enforce some additional requirements on the tree decomposition. For a set $S \subseteq \operatorname{vars}(\varphi)$ we let a *labeled* tree decomposition (T, χ, δ) of φ be a tree decomposition together with a mapping $\delta\colon V(T) \to S \cup \{\epsilon\}$ for a special empty symbol $\epsilon \notin \operatorname{vars}(\varphi)$ such that:

Unique Label For every $v \in S$ there is *exactly one* bag $t \in V(T)$ with $\delta(t) = v$.

Well Behaved For all $t \in V(T)$ with $\delta(t) \neq \epsilon$ we have $\delta(t) \in \chi(t)$ and t has exactly one child in T.

It is easy to transform a tree decomposition into a labeled one without increasing its width by coping bags as needed. Unless stated otherwise, in this article the set S will always be the set of variables involved in the cardinality constraint.

2.2 Logic Programming

While the focus of this article lies on propositional formulas and encodings therein, we realize our implementation in Section 5 using *logic programs* under *stable model semantics* (namely *Answer Set Programming,* ASP) (Gelfond and Lifschitz 1991) to describe and produce the *encodings* of the cardinality constraints. Conceptually, ground ASP is like SAT, but we require a justification for every atom that is derived (set to true). We refer the interested reader to the textbook by Lifschitz (2019). Current ASP systems (Gebser et al. 2012; Kaminski and Schaub 2021) offer a rich first-order like non-ground language to model and solve problems using advanced modeling constructs, aggregates, and theory reasoning. The use of first-order variables enables compact representations of encodings without the need for direct constraint formulations in propositional logic.

Example 1. *Recall the odd cycle transversal problem. Assume that a graph is given via a unary* `node` and a binary `edge` predicate, and let `limit` be the number of jokers. A possible ASP encoding is given in the following box, based on the ASP-Core-2 syntax (Calimeri et al. 2020).

```
% Every node gets 1 of 3 colors.
1{red(V); blue(V); joker(V)}1 :- node(V).
% No color collision (expect for jokers).
⊥ :- edge(U,V), red(U),  red(V).
⊥ :- edge(U,V), blue(U), blue(V).
% At most K jokers. This generates binom(n,K
    ) rules.
⊥ :- #count{V: joker(V)}>K, limit(K).
```

Readers unfamiliar with ASP can skip the implementation details in Section 5, as all the technical details are independently presented in Section 4 and analyzed in Section 6.

3 Study of Existing Encodings

In this section we will briefly review commonly used encodings for cardinality constraints and analyze whether they are parsimonious and their impact on the formulas treewidth. At the end of the description of every encoding, we added a small box to summarize our findings (name of the encoding, whether or not it is parsimonious, and the asymptotic worst-case increase on the formulas treewidth):

Encoding	Parsimony	Treewidth

An encoding is not necessarily well-defined and often the term "encoding" refers to a family of encodings (for instance, their are various ways to implement a sorting network). In such cases, we refer to the implementation in PySAT (Ignatiev, Morgado, and Marques-Silva 2018).

3.1 Naive (pairwise, binomial)

This approach has been presented in the introduction. It is easy to see that in the worst case, the naive encoding (Roussel and Manquinho 2021) can increase the treewidth by *the number of variables* in the cardinality constraint. The reason is that the pairwise comparisons of n variables forms a size-n clique, which has treewidth n.

Naive	✓	$\Omega(n)$

3.2 Sequential Counter

The sequential counter goes back to Sinz (2005) and exists in different variants and refinements, see, e.g., (Wynn 2018). The basic idea is to keep track of the count in sequential order. To this end, we will use *auxiliary variables* of the form $s_{i,j}$ which indicates that up to the i-th element, we already observed (counted) j true elements in $\{x_1, \ldots, x_i\}$. Initially, the counter can be at most 1, so $\neg s_{i,j}$ for $j > 1$:

$$\bigwedge_{1<j\leq k} \neg s_{1,j}. \tag{1}$$

Setting x_i increases the counter by 1:

$$\bigwedge_{1\leq i<n} (\neg x_i \vee s_{i,1}). \tag{2}$$

$$\bigwedge_{1<i<n} \bigwedge_{1<j\leq k} (\neg x_i \vee \neg s_{i-1,j-1}, \vee s_{i,j}). \tag{3}$$

We need to copy the value of the counter from step $i-1$ to i:

$$\bigwedge_{1<i<n} \bigwedge_{1\leq j\leq k} (\neg s_{i-1,j} \vee s_{i,j}). \tag{4}$$

Realization of the cardinality constraint:

$$\bigwedge_{1<i\leq n} (\neg x_i \vee \neg s_{i-1,k}). \tag{5}$$

Theorem 1. *The sequential counter encoding is not parsimonious, as it overcounts.*

Proof. The encoding requires us to increase the count if we pick an element x_i. However, in any step $1 < i < k$ we can also just increase the count without seeing a true literal. □

Without taking care of the sequential ordering, this implementation can increase the treewidth up to $\Omega(n)$.

Theorem 2. *Adding a k-cardinality sequential counter to a formula with n variables can increase the treewidth to $\Omega(n)$.*

Proof. We will assume that the primal graph G_φ of φ is a cycle. The formula could, for instance, be a set of implications: $x_1 \to x_2$, $x_2 \to x_3$, …, $x_{n-1} \to x_n$, and $x_n \to x_1$.

Any random 3-regular graph on n vertices has treewidth $\Omega(n)$ with high probability, see for instance Prop. 17 in Mehta and Reichman (2022) and (Friedman 2003; Alon and Milman 1985). Let us call this event E_t, i.e., $\Pr[E_t] \to 1$ for $n \to \infty$.

Furthermore, such random 3-regular graphs are Hamiltonian with high probability as well, see Thm. 1 in Robinson and Wormald (1994). Let us call this event E_h. Since both E_t and E_h individually have high probability, we have:

$$\begin{aligned}\Pr[E_t \cap E_h] &= 1 - \Pr[\neg(E_t \cup E_h)] \\ &\geq 1 - (\Pr[\neg E_t] + \Pr[\neg E_h]),\end{aligned}$$

as $\Pr[\neg E_t \cup \neg E_h] \leq \Pr[\neg E_t] + \Pr[\neg E_h]$. Hence, we obtain $\Pr[E_t \cap E_h] \to 1$ for $n \to \infty$, as $\Pr[\neg E_t] \to 0$ and $\Pr[\neg E_h] \to 0$. Consequently, by constructing a random 3-regular graph G, we eventually obtain one that has treewidth $\Omega(n)$ and is Hamiltonian. The Hamiltonian cycle will be the primal graph G_φ of our formula φ as sketched above. Clearly, this graph has treewidth 2. If we remove the edges of G_φ from G, we obtain a perfect matching M since G is 3-regular. Such a matching can be seen as a partial order, and any sequential counter necessarily introduces such an ordering to φ. Hence, it may add M to G_φ resulting back in G and thereby increasing the treewidth from 2 to $\Omega(n)$. □

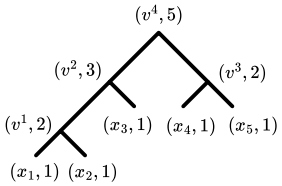


Figure 3: Binary tree of nodes (v, m) for the totalizer encoding of a set $S = \{x_1, x_2, x_3, x_4, x_5\}$ and their corresponding maximal counts m. The tree is balanced, the root corresponds to S, and leaves are singletons. Intuitively, the encoding precisely counts for each node by means of exact child node counts.

We will see later, in Section 4, how we can guarantee a treewidth increase up to k, which we then improve to $\log(k)$.

Sequential Counter	✗	$\Omega(n)$

3.3 Sorting Networks

Another common approach to implement cardinality constraints are *sorting networks* (Batcher 1968; Abío et al. 2013). The high-level idea is to encode a sorting algorithm that sorts the variables $x_1, \ldots, x_n$ into new auxiliary variables $y_1 \ldots, y_n$. Then to express, say, $\sum_{i=1}^{n} x_i \geq c$ we just have to add the constraint y_{n-c}. Unfortunately, a sorting network inherently solves all the cardinality constraints for all $c \in \{1, \ldots, n\}$ at once and, thus, any such encoding needs to have treewidth $\Omega(\log n)$ by Corollary 1. Similar arguments hold for cardinality networks (Asín et al. 2009).

It is not trivial to make a statement about whether this encoding is parsimonious, as it describes a family of encodings that could be implemented in various ways. Conceptually, the sorting function implemented by the network is total and, hence, these encodings are in theory parsimonious. However, we empirically observed that, for instance, the implementation of sorting networks in PySAT (Ignatiev, Morgado, and Marques-Silva 2018) is *not* parsimonious.

Sorting Networks	✓(✗)	$\Omega(\log n)$

3.4 Totalizer (Tree based, Bailleux & Boufkhad)

This encoding goes back to the work of Bailleux and Boufkhad (2003) and was improved in various ways for applications in MAX-SAT, e.g., by Ogawa et al. (2013) and Morgado, Ignatiev, and Marques-Silva (2014). The idea is to iteratively split the variables into two sets of roughly equal size, so one branch for a node with m elements considers $\lfloor \frac{m}{2} \rfloor$ elements and the other branch concerns the remaining $m - \lfloor \frac{m}{2} \rfloor$ elements. The leaf of these nodes contain original literals in S, as depicted in Figure 3.

Then, using such a binary tree with (v^r, m) being any non-leaf node with child nodes (v^i, m_i) and (v^j, m_j), we define $\geq (\alpha, \beta, \sigma)$ by the implication $(v^i_\alpha \wedge v^j_\beta) \to v^r_\sigma$ and $\leq (\alpha, \beta, \sigma)$ as $v^r_{\sigma+1} \to (v^i_{\alpha+1} \vee v^j_{\beta+1})$. It remains to define the clauses for any node (v_r, m):

$$\bigwedge_{\substack{0\leq\alpha\leq m_1, 0\leq\beta\leq m_2, \\ 0\leq\sigma\leq m, \alpha+\beta=\sigma}} [\geq (\alpha, \beta, \sigma)] \wedge [\leq (\alpha, \beta, \sigma)]. \tag{6}$$

Example 2. *Recall Figure 3 and note that for the leaf nodes (elements in S) no clauses are required, each element $x_i \in S$ already indicates whether among $\{x_i\}$ the count is 0 or 1. Then, the clauses for node $(v^1, 2)$ are:*

- $\sigma{=}0$*:* $(x_1 \vee x_2 \vee v_0^1) \wedge (\neg v_1^1 \vee x_1 \vee x_2)$,
- $\sigma{=}1$*:* $(\neg x_1 \vee x_2 \vee v_1^1) \wedge (\neg v_2^1 \vee x_1)$ *and* $(x_1 \vee \neg x_2 \vee v_1^1) \wedge (\neg v_2^1 \vee x_2)$,
- $\sigma{=}2$*:* $(\neg x_1 \vee \neg x_2 \vee v_2^1)$.

Note that for $\sigma{=}0$ and $\sigma{=}2$ there is only a single possibility among $\{x_1, x_2\}$; for $\sigma{=}1$ we can either choose x_1 or x_2.

The totalizer is one of the selected encodings (next to sorting networks) that is parsimonious in its original form.

Theorem 3. *The totalizer encoding is parsimonious, i.e., correct and complete, as it does not overcount.*

Proof. For each node (v^i, m) of the binary tree, the count can go from 0 to m and the totalizer encoding ensures that precisely one variable among $\{v_0^i, \ldots, v_m^i\}$ is true. □

In contrast to the theorem, we empirically observed that the implementation of the encoding in PySAT is *not* parsimonious. The input's structure is not preserved here either.

Corollary 2. *Adding a k-cardinality totalizer constraint to a formula with n variables increases the treewidth up to $\Omega(n)$.*

Proof. We utilize the proof of Theorem 2, where we let n be a power of 2, so the matching M relates 2 siblings of Figure 3 (encoding adds more auxiliary variables). □

Totalizer	✓(✗)	$\Omega(n)$

4 Parsimony and Structure-Awareness

In this section we develop encodings for cardinality constraints that are parsimonious *and* structure-aware. The idea underlying these encodings is that we *guide* the counter along a given tree decomposition.

4.1 Structure-Aware Sequential Counter

Initially, for the leaf nodes of T, the counter is 0:

$$\bigwedge_{t \in V(T): |\operatorname{children}(t,T)|=0, 0<j<k} \neg s_{t,j} \wedge s_{t,0}. \tag{7}$$

Setting x sets the counter of node t with $\delta(t) = x$ to $i{+}1$, where i is the counter at child node $\{t'\} = \operatorname{children}(t, T)$:

$$\bigwedge_{0 \le i < k} s_{t,i+1} \leftrightarrow (s_{t',i} \wedge x), \tag{8}$$

$$\bigwedge_{0 \le i < k} s_{t,i} \leftrightarrow (s_{t',i} \wedge \neg x). \tag{9}$$

We must not overcount for any element in the sequence:

$$s_{t',k} \rightarrow \neg x. \tag{10}$$

Further, for unlabeled TD nodes $t \in V(T)$, i.e., $\delta(t) = \epsilon$, we set the counter for t with $\operatorname{children}(t, T) = \{t_1, \ldots, t_o\}$ to i, if all child counters sum up to i:

$$\bigwedge_{0 \le i \le k} s_{t,i} \leftrightarrow \bigvee_{\substack{i_1,\ldots,i_o \mid 0 \le i_u \le k, \\ 1 \le u \le o, i = i_1 + \ldots + i_o}} (s_{t_1,i_1} \wedge \ldots \wedge s_{t_o,i_o}). \tag{11}$$

We must not overcount for any decomposition node:

$$\bigwedge_{\substack{i_1,\ldots,i_o \mid 0 \le i_u \le k, \\ 1 \le u \le o, k < i_1 + \ldots + i_o}} \neg s_{t_1,i_1} \vee \ldots \vee \neg s_{t_o,i_o}. \tag{12}$$

Lemma 1. *The Structure-Aware Sequential Counter is correct and complete.*

Proof. Correctness follows by the fact that the cardinality constraint holds iff the constructed encoding is satisfiable. The encoding ensures that for every assignment of S, there is precisely one unique assignment over the variables $s_{t,j}$. □

4.2 Adaptation for CNF

Observe that the Formulas (7)–(12) can easily be turned into CNF. However, for Formula (11) we require auxiliary variables, whereby a direct conversion would be in $O(k^n)$. We can resolve this, by adding auxiliary variables for every disjunct in Formula (11). We assume any arbitrary total order $\prec$ among tuples of the form $p = (i_1, \ldots, i_o)$ used in the disjunction. Then, for each pair $(i_1, \ldots, i_o)$, we define:

$$a_p \leftrightarrow (s_{t_1,i_1} \wedge \ldots \wedge s_{t_o,i_o}). \tag{13}$$

We link tuples as in $\prec$, where for the $\prec$-largest element p' and every successor relationship $p_i \prec p_{i+1}$ we construct

$$a_{\succeq p'} \leftrightarrow a_{p'} \qquad \text{and} \qquad a_{\succeq p_i} \leftrightarrow a_{p_i} \vee a_{\succeq p_{i+1}}. \tag{14}$$

Finally, we replace Formula (11) for the smallest pair p_1 by:

$$\bigwedge_{0 \le i \le k} s_{t,i} \leftrightarrow a_{\succeq p_1}. \tag{15}$$

Theorem 4. *Let φ be a formula and $\mathcal{T} = (T, \chi, \delta)$ be a width-w tree decomposition of G_φ. Then, a sequential k-cardinality constraint increases the width by at most $3k{+}6$.*

Proof. Without loss of generality, we assume that all bags have at most 2 children, which can be achieved by copying bags (Bodlaender and Kloks 1996). From $\mathcal{T}$ we construct a TD $\mathcal{T}' = (T', \chi')$ of $G_{\varphi'}$, where T' is constructed from T by replacing each node with $|\operatorname{children}(t, T)| > 1$ by a path whose length corresponds to the number of tuples of the form $p = (i_1, \ldots, i_o)$ in the disjunction of Formula (11). Define for every $t \in V(T)$ with $|\operatorname{children}(t, T)| \le 1$:

$$\chi'(t) = \chi(t) \cup \{s_{t,j} \mid 0 \le j \le k\} \cup \{s_{t',j} \mid 0 \le j \le k, t' \in \operatorname{children}(t, T)\}.$$

For t with $|\operatorname{children}(t, T)| > 1$, let the ith node of the path in T' be t_i and let $\prec$ be a total ordering among tuples. Define

$$\chi'(t_i) = \chi(t) \cup \{s_{t,j} \mid 0 \le j \le k\} \cup \{s_{t',j} \mid 0 \le j \le k, t' \in \operatorname{children}(t, T)\} \cup \{a_{p_i}, a_{p_{\succeq i}}\} \cup \{a_{\succeq p_{i+1}} \mid p_i \prec p_{i+1}\}.$$

Indeed, all clauses are covered and the decomposition is connected. Furthermore, $|\chi'(t)| \le |\chi(t)| + 3(k{+}1){+}3$. □

SAW Sequential	✓	$\Omega(w + k)$

4.3 Theoretical Limits and Lower Bounds

With a sequential *unary* counter, we can not avoid an increase of the treewidth up to k, as variables $s_{i,j}$ form a grid, see Figure 4.

Theorem 5. *A k-cardinality constraint over n variables using a structure-aware sequential counter can increases the treewidth by $\Omega(\min(n{-}k, k + 1))$.*

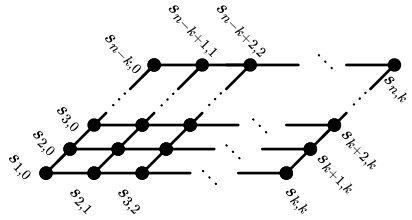


Figure 4: A sequential counter keeps track of increasing some count $j \leq k$ if an element x_i is chosen ($i \leq n$), otherwise existing counts need to be propagated. Structure contains a (skewed) $(n-k) \times (k+1)$ grid, which has treewidth $\geq \min(n-k, k+1)$.

Proof. An encoding by means of a sequential counter assumes that if we set x_i and if $s_{i-1,j-1}$ holds,then we require to set $s_{i,j}$. Consequently, we require links between $s_{i-1,j-1}$ and $s_{i,j}$ for any $1 < i \leq n$ and $1 \leq j \leq k$. Further, we need links between $s_{i-1,j}$ and $s_{i,j}$, as we don't necessarily set the i-th element x_i. However, as visualized in Figure 4, this induces a skewed grid, assuming the i-th column concerns $s_{i,*}$ (up to $i \leq n$ elements) and the j-th row is $s_{*,j}$ with $j \leq k$ (up to cardinality k). Since a $(n-k) \times (k+1)$ grid has treewidth at least $\min((n-k),(k+1))$ (Diestel 2012, Chaper 12), the treewidth of the formula to which we add the constraint may be increased by this value. □

Combining Theorems 4 and 5 implies that the structure-aware sequential counter with bound k executed using a width-w tree decomposition yields in general a formula of treewidth $\Theta(w+k)$. While this is already better than the bounds of of Section 3 (which increase the treewidth in dependence of the number of variables n), it does not hit the theoretical lower bound of $\Omega(w+\log(k))$ implied by Fact 1.

4.4 Logarithmic Treewidth via Binary Counter

We improve the dependency on k by using a *binary* counter instead of a sequential one. The encoding works similarly to Formulas (7)–(12), where every occurrence of $s_{t,j}$ is replaced by the binary encoding of j via variables $b_t^{j'}$ for $0 \leq j' < \lceil \log(k+1) \rceil$. To access the bit representation of a number n, we let $[n]_t$ *be a set of literals* over bits $b_t^{j'}$ that corresponds to the *bit representation of* n *for node* t. Initially, for leaf nodes of T we set the counters to 0:

$$\bigwedge_{t \in V(T): |\operatorname{children}(t,T)|=0, 0 \leq j < \lceil \log(k+1) \rceil} \neg b_{t,j}. \tag{16}$$

Setting x increments in node $t \in V(T)$ with $\delta(t)=x$ the counter at child node $\{t'\}=\operatorname{children}(t,T)$. This requires that bits flip from the least significant 1-bit to the first 0-bit:

$$\bigwedge_{0 \leq i < \lceil \log(k+1) \rceil} (x \wedge \bigwedge_{0 \leq i' < i} b_{t',i'}) \rightarrow (b_{t,i} \leftrightarrow \neg b_{t',i}). \tag{17}$$

From this 0-bit on to the most significant bit, we copy bits:

$$\bigwedge_{0 \leq i < \lceil \log(k+1) \rceil} (x \wedge \bigvee_{0 \leq i' < i} \neg b_{t',i'}) \rightarrow (b_{t,i} \leftrightarrow b_{t',i}). \tag{18}$$

If x is not set, we also copy bits (same counter):

$$\bigwedge_{0 \leq i < \lceil \log(k+1) \rceil} \neg x \rightarrow (b_{t,i} \leftrightarrow b_{t',i}). \tag{19}$$

We must not overcount for any element in the sequence:

$$(\bigwedge_{l_{t',i} \in [k]_{t'}} l_{t',i}) \rightarrow \neg x. \tag{20}$$

For t with $\delta(t) = \epsilon$ and $\operatorname{children}(t,T) = \{t_1, \ldots, t_o\}$, we set the counter to i if all child counters sum up to i:

$$\bigwedge_{0 \leq i \leq k, l_{t,b} \in [i]_t} l_{t,b} \leftrightarrow \bigvee_{\substack{i_1,\ldots,i_o | 0 \leq i_u \leq k, \\ 1 \leq u \leq o, i = i_1 + \ldots + i_o}} \bigwedge_{\substack{l_{t_1,b_1} \in [i_1]_{t_1}, \ldots, \\ l_{t_o,b_o} \in [i_o]_{t_o}}} (l_{t_1,b_1} \wedge \ldots \wedge l_{t_o,b_o}). \tag{21}$$

We must not overcount for any decomposition node:

$$\bigwedge_{\substack{i_1,\ldots,i_o | 0 \leq i_u \leq k, \\ 1 \leq u \leq o, k < i_1 + \ldots + i_o}} \bigvee_{l_{t_1,b_1} \in [i_1]_{t_1}, \ldots, l_{t_o,b_o} \in [i_o]_{t_o}} \neg l_{t_1,b_1} \vee \ldots \vee \neg l_{t_o,b_o}. \tag{22}$$

When converting to CNF, the disjunction of Formula (21) needs to be split using auxiliary variables. This works analogously to Section 4.2. Consequently, we obtain:

Theorem 6. *Let φ be a formula and $\mathcal{T} = (T, \chi, \delta)$ be a width-w tree decomposition of it. A k-cardinality constraint over n variables using a structure-aware binary counter increases the treewidth by at most $3\log(k+1)+3$.*

Proof. The proof works similarly to Theorem 4. From $\mathcal{T}$ we construct a TD $\mathcal{T}' = (T', \chi')$ of $G_{\varphi'}$, where we obtain T' from T by replacing nodes t with $|\operatorname{children}(t,T)| > 1$ by a path. For every node t with $|\operatorname{children}(t,T)| \leq 1$ define:

$$\begin{aligned} \chi'(t) = \chi(t) \cup \{b_{t,j} \mid 0 \leq j \leq \lceil \log(k+1) \rceil\} \\ \cup \{b_{t',j} \mid 0 \leq j \leq \lceil \log(k+1) \rceil, \\ t' \in \operatorname{children}(t,T)\}. \end{aligned}$$

For all $t \in V(T)$ with $|\operatorname{children}(t,T)| > 1$, let for the ith node t_i of the copied path:

$$\begin{aligned} \chi'(t) = \chi(t) \cup \{b_{t,j} \mid 0 \leq j \leq \lceil \log(k+1) \rceil\} \\ \cup \{b_{t',j} \mid 0 \leq j \leq \lceil \log(k+1) \rceil, \\ t' \in \operatorname{children}(t,T)\} \\ \cup \{a_{p_i}, a_{p_{\succeq i}}\} \cup \{a_{\succeq p_{i+1}} \mid p_i \prec p_{i+1}\}. \end{aligned}$$

Then, $|\chi'(t)| \leq |\chi(t)| + 3\lceil \log(k+1) \rceil + 3$, as desired. □

SAW Bin-Sequent.	✓	$\Omega(w+\log(k))$

4.5 Structure-Aware Totalizer

The next encoding we present is a structure-aware version of the totalizer. Essentially, the main difference between a totalizer and a sequential counter is that the sequential counter adds up literals in a linear way (we could say, "a long a path"), while the totalizer adds the literals using divide-and-conquer (say, "a long a binary tree"). Since the structure-aware version of the sequential counter already distributes the linear counter a long a tree, what should a structure-aware totalizer be? Of course, we need to distribute the binary divide-and-conquer tree across the given tree decomposition – without increasing its width! Note that it is, at first, not obvious how to archive this, as the given tree decomposition could, for instance, be a path.

Fortunately, we can resolve this issue purely on the graph-theoretic layer while reusing our structure-aware sequential

counter. A result from structural graph theory states that any tree decomposition of width w can be transformed into a binary, balanced decomposition of with at most $4w + 3$ (Elberfeld, Jakoby, and Tantau 2010). Binary here means that every node has at most two children, and balanced means that the length of the longest path from the root to one of the leaves is bounded by $O(\log n)$. We obtain a structure-aware totalizer by first applying the just mentioned transformation on the given tree decomposition, followed by the structure-aware encodings discussed in the previous section – yielding the following result as a special case of Theorem 4.

Corollary 3. *Let φ be a formula and $\mathcal{T} = (T, \chi, \delta)$ be a width-w tree decomposition of it. A k-cardinality constraint over n variables using a structure-aware totalizer increases the treewidth to at most $4w+3k+9$.*

SAW Totalizer	✓	$\Omega(w + k)$

As in Section 4.4, we can also use binary counters to obtain the following special case of Theorem 6.

Corollary 4. *Let φ be a formula and $\mathcal{T} = (T, \chi, \delta)$ be a width-w tree decomposition of it. A k-cardinality constraint over n variables using a structure-aware binary totalizer increases the treewidth to at most $4w+3\log(k+1)+6$.*

SAW Bin-Totalizer	✓	$\Omega(w + \log(k))$

Summarizing this section, we developed structure-aware versions of the sequential and binary counter. The shape of the tree decomposition (T, χ) used to produce the encoding determines its behavior: If the decomposition is trivial (e.g., $V(T) = \{t\}$ and $\chi(t) = V(G)$), our encoding is equivalent to the naive encoding; if T is a path, our encodings are precisely the sequential counter (or its binary version); and if T is a binary, balanced tree, our encodings reassemble the totalizer encoding – see Figure 5. Of course, the structure-aware encodings also work for all other shapes of tree decompositions (which do not correspond to classical encodings) and always preserve the input's structure.

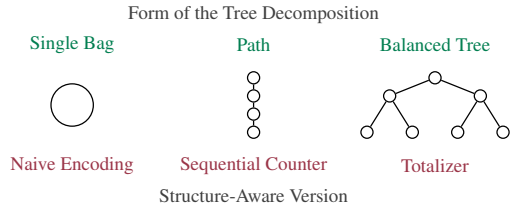


Figure 5: Visualization of how the tree decomposition defines structure-awareness for the used cardinality constraint encoding.

5 Prototypical Implementation

Since the encoding of structure-aware constraints can be cumbersome, we decided to utilize logic programming (ASP) to generate formulas in CNF. The advantage of this formalism for generating CNF formulas is that we obtain a compact and easy-to-read encoding, without losing #SAT performance. In more detail, our toolchain works as follows: First, we generate a tree decomposition with the `htd` decomposition library (Abseher, Musliu, and Woltran 2017), which is based on efficient heuristics and provides a variety of additional options for obtaining normalized representations. We add the given formula and the resulting tree decomposition to a fact representation and ground using *gringo* (Kaminski and Schaub 2021). After grounding, we simplify and pre-evaluate the obtained grounded rules via the ASP solver clingo (Gebser et al. 2012). Then, we use existing translation tools from ASP to SAT (Janhunen 2006; Bomanson and Janhunen 2013). The complete command line carried out in the process is:

```
gringo --output smodels | clasp --pre=
    smodels | lp2normal2 | lp2lp2 | lp2sat
```

5.1 ASP Encodings

We provide both the sequential and the binary encoding by means of logic programs. This avoids cumbersome SAT generators and therefore allows for easy maintainability and readability in first-order like representation. However, one has to be careful when it comes to avoiding substantial blowups due to grounding, which is particularly true when utilizing aggregates. While it is known that grounding has to cause an exponential blowup in general (Gebser et al. 2012), practical encodings are of bounded predicate arity (Eiter et al. 2007) and without substantial blowup in rule sizes.

Independent of the implementation, we formalize cardinality constraints by the predicates `card` and `limit`. The first is ternary and specifies that for a cardinality constraint C a term T shall be counted as mode M, whereas `limit` is binary and gives for C the highest count K we may reach among all (term, mode) tuples for C specified by `card`.

Example 3. *If we want to add a single cardinality constraint C over a set L of literals (given as signed integer) on top of a logical formula φ, we specify the constraint as follows: The cardinality constraint has an upper limit k:* `card(C,k).`

For every literal $l \in L$, we define that it is contained in the cardinality constraint, if it is assigned accordingly. To this end, we assume that variable assignments are decided by a predicate `true`(l) that holds the set of true literals:

```
card(C,l,neg) :- l<0, true(l).
card(C,l,pos) :- l>0, true(l).
```

The sequential encoding is established by means of the following two non-ground ASP rules.

```
cnt(C,T,E) :- node(T),limit(C,K),E<=K,
 E=#sum{1,M,X: label(T,X),card(C,X,M) ;
       D,t,T': cnt(C,T',D),child(T,T')}.
⊥ :- node(T),limit(C,_),¬cnt(C,T,_).
```

Intuitively, the first rule sums up the counts for child nodes of T as well as the number of elements of the cardinality constraints that are in the labeling for T. Then, it stores the result if it is $\leq k$ for the cardinality bound k. The second rule prohibits counts that are $> k$, as no sum $\leq k$ has been derived for constraint C and node T.

The binary encoding is slightly more involved, requiring careful bit twiddling within ASP. Here, the second rule stays

the same as above, while we rewrite the first rule such that the `cnt` predicate now indicates that the P-th bit is set to 1.

```
cnt(C,T,P) :- node(T),limit(C,K),E<=K,
 E=#sum{1,M,X: label(T,X),card(C,X,M) ;
   2**P',t,T': cnt(C,T',P'),child(T,T'),
     0<=P'<=P},(E/(2**P))&1=1,bit(C,P).
```

6 Experimental Evaluation

To evaluate our structure-aware cardinality constraints, we need a benchmark set that (1) contains instances of low treewidth, and (2) has a counting objective that must be scalable independent of the instances treewidth.

An interesting combinatorial problem that fits these needs is the *odd cycle transversal* problem (OCT), which we already used as an example in the introduction. The input to this problem is an undirected graph G and a number k, and the question is whether we can delete k vertices from G such that the remaining graph is bipartite (i.e., such that all odd cycles traverse through the k selected nodes, hence the name). Besides many interesting applications in computational biology (Hüffner 2009) and quantum computing (Goodrich, Sullivan, and Humble 2018; Goodrich, Horton, and Sullivan 2021), the problem has in particular the property that the minimum size of an OCT in G is independent of the treewidth of G. This can be seen as the family of star graphs in which we replace every leaf with a triangle has treewidth 2, but an unbounded OCT; and the other way around since the family of fully-connected bipartite graphs has unbounded treewidth, but an OCT of size 0. For our benchmark set, we define the family of star-chain graphs S_n as being a chain of n stars, each having $n-1$ leaves. Let then $S_{n,k}$ for $k \in \{1,\dots,n-1\}$ be the graph obtained by taking S_n and replacing k leaves of every star with a triangle, see Table 1 for some examples.

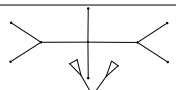
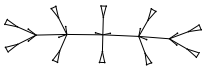
Graph	Image	OCT	treewidth
S_3		0	1
$S_{3,2}$		6	2
S_5		0	1
$S_{5,2}$		10	2

Table 1: Four examples of the chain-star family and corresponding minimum size of an odd cycle transversal and the treewidth.

Trivially, the treewidth of every $S_{n,k}$ is at most 2 and the minimum size of an odd cycle transversal is kn, since each triangle must contribute a node to the OCT. Furthermore, the number of minimum-size odd cycle transversals in $S_{n,k}$ is exactly $2 \cdot 4^{kn}$. Our *benchmark set* consists of 190 graphs $S_{n,k}$ with $n \in \{2,\dots,20\}$ and $k \in \{1,\dots,n-1\}$.

6.1 Benchmark Scenario

We run our experiments on a server equipped with two `AMD EPYC 7702` 64-core processors operating at a maximum clock speed of `3353.5149 MHz`. The server has a total of `504 GB` of RAM and runs `Arch Linux` with kernel version `6.13.5-arch1-1`. For each graph $S_{n,k}$ in the benchmark set we counted the number of odd cycle transversals of size kn, i.e., the ones of minimum size. In the experiments reported here we used the model counter `d4` (Lagniez and Marquis 2017) as back end; we obtained similar figures (not reported in this article) using the counters `SharpSAT-TD` (Korhonen and Järvisalo 2023) and `GPMC` (Suzuki, Hashimoto, and Sakai 2017). We used the PySAT library (Ignatiev, Morgado, and Marques-Silva 2018) to encode existing cardinality constraints as discussed in Section 3, and well-known ASP tools (Janhunen 2006; Bomanson and Janhunen 2013) to generate SAT encodings of structure-aware cardinality constraints. For each instance we measured the complete time of the run, that is, including time to read and write the instance, to produce the encoding, and for running the model counter. Each encoding had a maximum time budget of 600 seconds per instance.

6.2 Experimental Results

In the light of **Q2**, we first analyzed the impact of the various encodings on the input's treewidth. Figure 6 shows for every instance the (heuristically computed) treewidth of the corresponding formula together with one of the cardinality constraints. As mentioned before, all formulas have treewidth 2 without the cardinality constraint, and the used heuristic (Abseher, Musliu, and Woltran 2017) found a tree decomposition of width at most 3 for all of them (red dashed line). The blue line shows the asymptotic bound guaranteed by our *structure-aware sequential counter*.

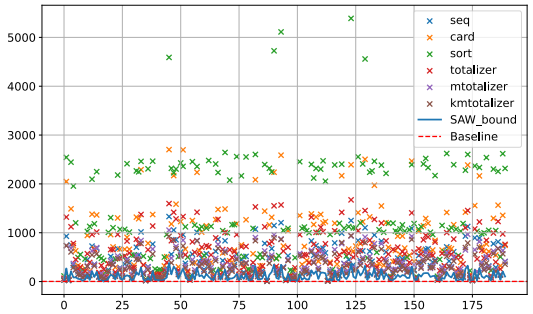


Figure 6: For each instance in the benchmark set (x-axis), the plot contains a cross for the treewidth of every encoding (y-axis). The blue line is the bound of our structure-aware sequential counter.

Two observations are relevant: First, there is no cross under the blue curve, which means *none* of the traditional encodings increases the inputs treewidth less than our encoding on *any* instance. Second, as predicted by our theoretical analysis in Section 3, the classical cardinality constraints can increase the treewidth quite dramatically – regularly by over 500, sometimes even by more than 2000.

We next compare the classical sequential counter to our

structure-aware version (SAW) that we introduced in Section 4.1. The *dots* in Figure 7 presents these results in a *scatter plot* that contains one dot for every instance, whereby the x-coordinate is the time (in seconds) that the structure-aware encoding needed, and the y coordinate the time required by the classical sequential counter.

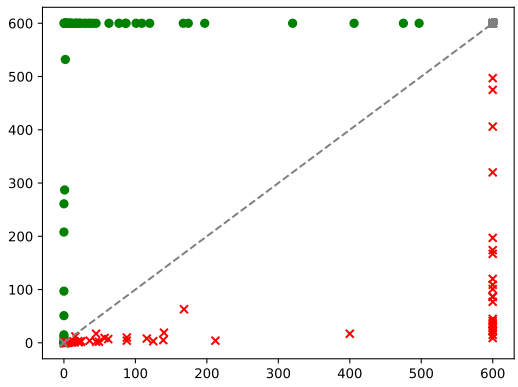


Figure 7: Dots: Comparison of a sequential counter (y-axis in seconds) with its structure-aware counter part (x-axis). A dot is green if the structure-aware encoding performs better on that instance, it would be red otherwise (no such instance). Crosses: Comparison of the structure-aware *binary* counter (x-axis) to the structure-aware *sequential* counter (y-axis).

We can observe that there is not a single instance within the benchmark set on which the structure-aware sequential counter performs worse than its classical counter part. To bring this result into perspective, we emphasis that the instances in the benchmark set are well-structured (the SAT formulas all have treewidth two without the cardinality constraint), i.e., counting a solution *without* the cardinality constraint should be easy. The insight that a cardinality constraint that is structure-aware performs noticeably better highlights the relevance of **Q2** and underlines that model counting is currently only feasible on structured instances.

The crosses in Figure 7 compare the binary version (Section 4.4) of our structure-aware counter with its sequential version. Here we observe that the binary version does *not* provide any additional improvement. We suspect this is the case due to the increased complexity in the encoding.

Figure 8 presents a *cumulative function plot* that compares the performance of our structure-aware encodings to all cardinality constraint encodings available in PySAT (including the ones that are not parsimonious). It reveals that the generated benchmark set is indeed very challenging as the best performing #SAT encoding (the sequential counter, shown in brown) is only able to solve about 12% of the instances. In contrast, our structure-aware sequential (green) and binary (red) encodings outperform all other encodings with a margin and solve 37% of the instances.

Another interesting finding is that counting via native answer set *enumeration* (violet) directly in ASP outperforms the route over #SAT if cardinality constraints are involved. This again highlights the importance of structure and **Q2** as the other encodings strip the structure from the formula.

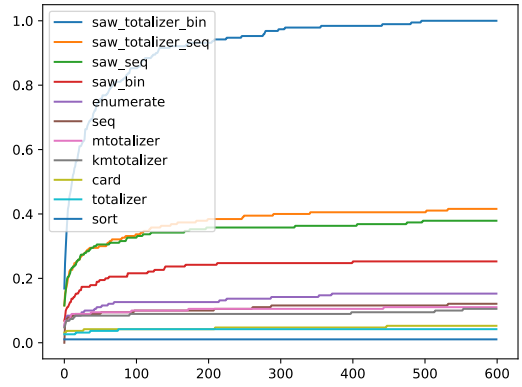


Figure 8: For each encoding the plot contains the percentage of instances that can be solved within the time defined by the x-axis.

For our final experiment, we compare to the *totalizer* version of our structure-aware encodings (Section 4.5). Recall that this is essentially the same encoding, but applied after we have transformed the given tree decomposition into a binary, balanced one (i.e., every node has at most 2 children and the depth of the decomposition is bounded by $O(\log n)$). We implemented a simple routine based on (Elberfeld, Jakoby, and Tantau 2010) that, given a width-w tree decomposition, outputs a binary, balanced decomposition of width at most $4w + 3$. Figure 8 contains our structure-aware sequential counter (green) and binary counter (red), as well as their totalizer versions (orange and blue). The totalizer version of the sequential counter performs slightly better than its normal version – the performance improvement we expected. The totalizer version of the binary counter, however, improves the performance more than we had expected – indeed, this version is able to solve all instances in the benchmark set within the given time limit. We suspect that this encoding combines the best of both worlds: The structure of the formula is only increased by a logarithmic factor, while used auxiliary variables are minimized.

7 Conclusion and Outlook

We studied cardinality constraints in the context of model counting and showed that none of the famous encodings is parsimonious and structure-aware. Based on this insight and since model counting is feasible on well-structured instances, we developed encodings that are both, parsimonious *and* structure-aware. Our experiments revealed that existing encodings eradicate the input's treewidth (Figure 6), while our encodings perform significantly better (Figure 8). This is strong indicator that *structure counts for counting*.

Our benchmarks were performed on handcrafted instances on which we have full control over the treewidth. Further research is needed to analyze the performance of structure-aware encodings on real world instances. Due to the magnitude of the performance improvement, structure-aware cardinality constraints may also turn out to be useful in areas where structure is less important, for instance in classical SAT or MAX-SAT. Since we described our encodings in ASP, they can also be used in this formalism.

Acknowledgments

This research was supported by the Austrian Science Fund (FWF) grant 10.55776/J4656.

References

Abío, I.; Nieuwenhuis, R.; Oliveras, A.; and Rodríguez-Carbonell, E. 2013. A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In Schulte, C., ed., *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, 80–96. Springer.

Abseher, M.; Musliu, N.; and Woltran, S. 2017. htd - A Free, Open-Source Framework for (Customized) Tree Decompositions and Beyond. In *14th International Conference on the Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, volume 10335 of *LNCS*, 376–386. Springer.

Alon, N., and Milman, V. D. 1985. lambda$_1$, Isoperimetric Inequalities for Graphs, and Superconcentrators. *J. Comb. Theory B* 38(1):73–88.

Asín, R.; Nieuwenhuis, R.; Oliveras, A.; and Rodríguez-Carbonell, E. 2009. Cardinality Networks and Their Applications. In Kullmann, O., ed., *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, 167–180. Springer.

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF Encoding of Boolean Cardinality Constraints. In Rossi, F., ed., *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, 108–122. Springer.

Batcher, K. E. 1968. Sorting Networks and Their Applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, volume 32 of *AFIPS Conference Proceedings*, 307–314. Thomson Book Company, Washington D.C.

Berre, D. L.; Marquis, P.; Mengel, S.; and Wallon, R. 2018. Pseudo-Boolean Constraints from a Knowledge Representation Perspective. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 1891–1897. ijcai.org.

Bianco, G. L.; Lorca, X.; Truchet, C.; and Pesant, G. 2019. Revisiting Counting Solutions for the Global Cardinality Constraint. *J. Artif. Intell. Res.* 66:411–441.

Bodlaender, H. L., and Kloks, T. 1996. Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs. *Journal of Algorithms* 21(2):358–402.

Bomanson, J., and Janhunen, T. 2013. Normalizing Cardinality Rules Using Merging and Sorting Constructions. In *LPNMR*, volume 8148 of *LNCS*, 187–199. Springer.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. ASP-Core-2 Input Language Format. *Theory Pract. Log. Program.* 20(2):294–309.

Chavira, M., and Darwiche, A. 2008. On Probabilistic Inference by Weighted Model Counting. *Artif. Intell.* 172(6-7):772–799.

Diestel, R. 2012. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer.

Eiter, T.; Faber, W.; Fink, M.; and Woltran, S. 2007. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artif. Intell.* 51(2-4):123–165.

Elberfeld, M.; Jakoby, A.; and Tantau, T. 2010. Logspace Versions of the Theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, 143–152. IEEE Computer Society.

Fichte, J. K.; Hecher, M.; Morak, M.; Thier, P.; and Woltran, S. 2023. Solving Projected Model Counting by Utilizing Treewidth and its Limits. *Artif. Intell.* 314:103810.

Fichte, J. K.; Hecher, M.; and Hamiti, F. 2021. The Model Counting Competition 2020. *ACM Journal of Experimental Algorithmics* 26(1):1–26.

Friedman, J. 2003. A Proof of Alon's Second Eigenvalue Conjecture. In Larmore, L. L., and Goemans, M. X., eds., *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, 720–724. ACM.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Gener. Comput.* 9(3/4):365–386.

Goodrich, T. D.; Horton, E.; and Sullivan, B. D. 2021. An Updated Experimental Evaluation of Graph Bipartization Methods. *ACM J. Exp. Algorithmics* 26:12:1–12:24.

Goodrich, T. D.; Sullivan, B. D.; and Humble, T. S. 2018. Optimizing Adiabatic Quantum Program Compilation Using a Graph-Theoretic Framework. *Quantum Inf. Process.* 17(5):118.

Hüffner, F. 2009. Algorithm Engineering for Optimal Graph Bipartization. *J. Graph Algorithms Appl.* 13(2):77–98.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, 428–437.

Janhunen, T. 2006. Some (In)Translatability Results for Normal Logic Programs and Propositional Theories. *Journal of Applied Non-Classical Logics* 16(1-2):35–86.

Jo, W., and Cho, H. 2025. DCC: Differentiable Cardinality Constraints for Partial Index Tracking. In Walsh, T.; Shah, J.; and Kolter, Z., eds., *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, 11264–11271. AAAI Press.

Kaminski, R., and Schaub, T. 2021. On the Foundations of Grounding in Answer Set Programming. *CoRR* abs/2108.04769.

Korhonen, T., and Järvisalo, M. 2023. SharpSAT-TD in Model Counting Competitions 2021-2023. *CoRR* abs/2308.15819.

Lagniez, J.-M., and Marquis, P. 2017. An Improved Decision-DNNF Compiler. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, 667–673. ijcai.org.

Leung, M., and Wang, J. 2022. Cardinality-Constrained Portfolio Selection Based on Collaborative Neurodynamic Optimization. *Neural Networks* 145:68–79.

Liang, Y., and Lin, G. 2024. Relaxed Method for Optimization Problems with Cardinality Constraints. *J. Glob. Optim.* 88(2):359–375.

Lifschitz, V. 2019. *Answer Set Programming*. Springer.

Malhotra, S., and Serafini, L. 2022. Weighted Model Counting in FO2 with Cardinality Constraints and Counting Quantifiers: A Closed Form Formula. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 5817–5824. AAAI Press.

Mehta, H., and Reichman, D. 2022. Local Treewidth of Random and Noisy Graphs with Applications to Stopping Contagion in Networks. In Chakrabarti, A., and Swamy, C., eds., *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, 7:1–7:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Morgado, A.; Ignatiev, A.; and Marques-Silva, J. 2014. MSCG: Robust Core-Guided MaxSAT Solving. *J. Satisf. Boolean Model. Comput.* 9(1):129–134.

Ogawa, T.; Liu, Y.; Hasegawa, R.; Koshimura, M.; and Fujita, H. 2013. Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, 9–17. IEEE Computer Society.

Olivé, A. 2007. *Conceptual Modeling of Information Systems*. Springer.

Papadimitriou, C. H. 2007. *Computational Complexity*. Academic Internet Publ.

Pichler, R.; Rümmele, S.; Szeider, S.; and Woltran, S. 2010. Tractable Answer-Set Programming with Weight Constraints: Bounded Treewidth Is not Enough. In Lin, F.; Sattler, U.; and Truszczynski, M., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press.

Robinson, R. W., and Wormald, N. C. 1994. Almost All Regular Graphs Are Hamiltonian. *Random Struct. Algorithms* 5(2):363–374.

Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier.

Roussel, O., and Manquinho, V. 2021. Pseudo-Boolean and Cardinality Constraints. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 1087–1129.

Russell, S., and Norvig, P. 2020. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

Samer, M., and Szeider, S. 2021. Fixed-Parameter Tractability. In *Handbook of Satisfiability - Second Edition*, Frontiers in Artificial Intelligence and Applications. IOS Press. 693–736.

Shaw, A., and Meel, K. S. 2024. Model Counting in the Wild. In Marquis, P.; Ortiz, M.; and Pagnucco, M., eds., *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*.

Sinz, C. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In van Beek, P., ed., *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, 827–831. Springer.

Suzuki, R.; Hashimoto, K.; and Sakai, M. 2017. Improvement of Projected Model-Counting Solver with Component Decomposition using SAT Solving in Components. Technical report, Technical report, JSAI Technical Report, SIG-FPAI-103-B506.

Tillmann, A. M.; Bienstock, D.; Lodi, A.; and Schwartz, A. 2024. Cardinality Minimization, Constraints, and Regularization: A Survey. *SIAM Rev.* 66(3):403–477.

Tóth, J., and Kuzelka, O. 2024. Complexity of Weighted First-Order Model Counting in the Two-Variable Fragment with Counting Quantifiers: A Bound to Beat. In Marquis, P.; Ortiz, M.; and Pagnucco, M., eds., *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*.

Wynn, E. 2018. A Comparison of Encodings for Cardinality Constraints in a SAT Solver. *CoRR* abs/1810.12975.

Zhang, Z.; Chen, W.; and Hu, X. 2023. A Knowledge-Based Constructive Estimation of Distribution Algorithm for Bi-Objective Portfolio Optimization with Cardinality Constraints. *Appl. Soft Comput.* 146:110652.