Algorithmica (2023) 85:584–609
https://doi.org/10.1007/s00453-022-01034-7

Faster Graph Coloring in Polynomial Space

Serge Gaspers[1] · Edward J. Lee[1]

Received: 12 August 2021 / Accepted: 23 August 2022 / Published online: 11 October 2022

Abstract
We present a polynomial-space algorithm that computes the number of independent sets of any input graph in time $O(1.1389^n)$ for graphs with maximum degree 3 and in time $O(1.2356^n)$ for general graphs, where n is the number of vertices in the input graph. Together with the inclusion-exclusion approach of Björklund, Husfeldt, and Koivisto [SIAM J. Comput. 2009], this leads to a faster polynomial-space algorithm for the graph coloring problem with running time $O(2.2356^n)$ as well as an exponential-space $O(1.2330^n)$ time algorithm for counting independent sets. Our main algorithm counts independent sets in graphs with maximum degree at most 3 and no vertex with three neighbors of degree 3. This polynomial-space algorithm is designed and analyzed using the recently introduced Separate, Measure and Conquer approach [Gaspers & Sorkin, ICALP 2015]. Using Wahlström's compound measure approach, this improvement in running time for small degree graphs is then bootstrapped to larger degrees, giving the improvement for general graphs. Combining both approaches leads to some inflexibility in choosing vertices to branch on for the small-degree cases, which we counter by structural graph properties.

Keywords Graph coloring · Exponential time algorithms · Analysis of algorithms · Branching algorithms · Counting independent sets

Mathematics Subject Classification F.2.2 Nonnumerical Algorithms and Problems · G.2.2 Graph Theory

✉ Edward J. Lee
e.lee@unsw.edu.au

Serge Gaspers
sergeg@cse.unsw.edu.au

[1] UNSW Sydney, Sydney, Australia

1 Introduction

Graph coloring is a central problem in discrete mathematics and computer science. In exponential time algorithmics [15], graph coloring is among the most well studied problems, and it is an archetypical partitioning problem. Given a graph G and an integer k, the problem is to determine whether the vertex set of G can be partitioned into k independent sets. Already in 1976, Lawler [25] designed a dynamic programming algorithm for graph coloring and upper bounded its running time by $O(2.4423^n)$, where n is the number of vertices of the input graph. This was the best running time for graph coloring for 25 years, when Eppstein [10] improved the running time to $O(2.4150^n)$ by using better bounds on the number of small maximal independent sets in a graph. Based on bounds on the number of maximal induced bipartite subgraphs and refined bounds on the number of size-constrained maximal independent sets, Byskov [7] improved the running time to $O(2.4023^n)$. An algorithm based on fast matrix multiplication by Björklund and Husfeldt [3] improved the running time to $O(2.3236^n)$. The current fastest algorithm for graph coloring, by Björklund et al. [2, 4] and, [24], is based on the principle of inclusion–exclusion and Yates' algorithm for the fast zeta transform. This breakthrough algorithm solves graph coloring in $O^*(2^n)$ time, where the O^*-notation is similar to the O-notation but ignores polynomial factors.

A significant drawback of the aforementioned algorithms is that they use exponential space. Often, the space bound is the same as the time bound, up to polynomial factors. This exponential space bound is undesirable [30], certainly for modern computing devices. Polynomial-space algorithms for graph coloring have been studied extensively as well with successive running times $O^*(n!)$ [8], $O((k/e)^n)$ (randomized) [11], $O((2+\log k)^n)$ [1], $O(5.283^n)$ [6], $O(2.4423^n)$ [3], and $O(2.2461^n)$ [4]. The latter algorithm is an inclusion–exclusion algorithm relying on a $O(1.2461^n)$ time algorithm [16] for computing the number of independent sets in a graph as a subroutine. Their method transforms any polynomial-space $O(c^n)$ time algorithm for counting independent sets into a polynomial space $O((1+c)^n)$ time algorithm for graph coloring. The running time bound for counting independent sets was subsequently improved by Fomin et al. [12] to $O(1.2431^n)$ and by Wahlström [29] to $O(1.2377^n)$. Wahlström's algorithm is the current fastest published algorithm for counting independent sets of a graph, it uses polynomial space, and it works for the more general problem of computing the number of maximum-weight satisfying assignments of a 2-CNF formula. For a reduction from counting independent sets to counting maximum-weight satisfying assignments of a 2-CNF formula where the number of variables equals the number of vertices, see [9].

We note that Junosza-Szaniawski and Tuczynski [23] present an algorithm for counting independent sets with running time $O(1.2369^n)$ in a technical report that also strives to disconnect low-degree graphs. For graphs with maximum degree 3 that have no degree-3 vertex with all neighbors of degree 3, they present a new algorithm with running time $2^{n_3/5+o(n)}$, where n_3 is the number of degree-3 vertices, and the overall running time improvement comes from plugging this result into Wahlström's [29] previously fastest algorithm for the problem. However, we note that the $2^{n_3/5+o(n)}$ running time for counting independent sets can easily be obtained from previous results. Namely, the problem of counting independent sets is a polynomial constraint

satisfaction problem (PCSP) with domain size 2, as shown in [27]. The algorithm of [20] for PCSPs preprocesses all degree-2 vertices, leaving a cubic graph on n_3 vertices that is solved in $2^{n/5+o(n)}$ time. Improving on this bound is challenging, and degree-3 vertices with all neighbors of degree 2 need special attention since branching on them affects the degree-3 vertices of the graph exactly the same way as for the much more general PCSP problem, whereas for other degree-3 vertices one can take advantage of the asymmetric nature of the typical independent set branching (i.e., we can delete the neighbors when counting the independent sets containing the vertex we branch on).
Our Results.

We present a polynomial-space algorithm computing the number of independent sets of any input graph G in time $O(1.2356^n)$, where n is the number of vertices of G. Our algorithm is a branching algorithm that works initially similarly as Wahlström's algorithm, where we slightly improve the analysis using potentials (as, e.g., in [19, 22, 28]) to amortize some of the worst branching cases with better ones. This algorithm uses a branching strategy that basically ensures that both the maximum degree and the average degree of the graph do not increase. This makes it possible to divide the analysis of the algorithm into sections depending on what local structures can still occur in the graph, use a separate measure for the analysis of each section, and combine these measures giving a compound (piecewise linear) measure for the analysis of the overall algorithm.

We also specifically design a subroutine to handle instances where the maximum degree is 3 and no vertex has three neighbors with degree 3, and is designed and analyzed using the recently introduced *Separate, Measure and Conquer* technique [20]. In this subroutine, the average degree of the graph is at most $8/3$. It computes a small balanced separator of the graph and prefers to branch on vertices in the separator, adjusting the separator as needed by the analysis, and reaping a huge benefit when the separator is exhausted and the resulting connected components can be handled independently. The Separate, Measure and Conquer technique helps to amortize this sudden gain with the analysis of the previous branchings, for an overall improvement of the running time.

Since using a separator restricts our choice in the vertices to branch on, we use the structure of the graph and its separation to upper bound the number of unfavorable branching situations and adapt our measure accordingly. Namely, the algorithm avoids branching on degree-3 vertices in the separator with all neighbors of degree 2 as long as possible, often rearranging the separator to avoid this case. In our analysis we can then upper bound the number of unfavorable branchings and give the central vertex involved in such a branching a special weight and role in the analysis. We call these vertices *spider vertices*. Our meticulous analysis of this subroutine upper bounds its running time by $O(1.0963^n)$. For graphs with maximum degree at most 3, we obtain a running time of $O(1.1389^n)$. This improvement for small degree graphs is bootstrapped, using Wahlström's compound measure analysis, to larger degrees, and gives a running time improvement to $O(1.2356^n)$ for counting independent sets of arbitrary graphs and to $O(2.2356^n)$ for graph coloring. Bootstrapping an exponential-space pathwidth-based $O(1.1225^n)$ time algorithm [14] for cubic graphs instead, we obtain an exponential-space algorithm for counting independent sets with running time $O(1.2330^n)$. A preliminary version of this paper appeared in the proceedings of COCOON 2017 [18].

2 Methods

Measure and Conquer. The analysis of our algorithm is based on the Measure and Conquer method [13]. A *measure* for a problem (or its instances) is a function from the set of all instances of the problem to the set of non-negative reals. Modern branching analyses often use a potential function as measure that gives a more fine-grained way of tracking the progress of a branching algorithm than a measure that is merely the number of vertices or edges of the graph. The following lemma is at the heart of our analysis. It generalizes a similar lemma from [19] to the treatment of subroutines. The proof of the lemma (see Lemma 2.6 in [17]) is a simple inductive argument. It uses a measure μ to upper bound the number of leaves of any search tree of an algorithm A, a measure η to upper bound the height of such a search tree, and a measure μ_B that upper bounds the running time of an algorithm B that algorithm A may call as a subroutine instead of branching.

Lemma 1 (Lemma 2.6 in [17]) *Let A be an algorithm for a problem P. Let B be an algorithm for a class $\mathcal{C}$ of instances of P, $c \geq 0$ and $r > 1$ be constants. The functions $\mu(\cdot)$, $\mu_B(\cdot)$, $\eta(\cdot)$ are measures for P such that for any input instance I from $\mathcal{C}$, $\mu_B(I) \leq \mu(I)$. If for any input instance I, A either solves P on $I \in \mathcal{C}$ by invoking B with running time $O(\eta(I)^{c+1} r^{\mu_B(I)})$, or reduces I to k instances $I_1, \ldots, I_k$, solves these recursively, and combines their solutions to solve I, using time $O(\eta(I)^c)$ for the reduction and combination steps (but not the recursive solves),*

$$(\forall i) \quad \eta(I_i) \leq \eta(I) - 1, \textit{ and} \tag{1}$$

$$\sum_{i=1}^{k} r^{\mu(I_i)} \leq r^{\mu(I)} \ , \tag{2}$$

then A solves any instance I in time $O(\eta(I)^{c+1} r^{\mu(I)})$.

When Algorithm A does not invoke Algorithm B, we have the usual Measure and Conquer analysis. Here, μ is used to upper bound the number of leaves of the search tree and deserves the most attention, while η is usually a polynomial measure that is used to upper bound the depth of the search tree. For handling subroutines, it is crucial that the measure does not increase when Algorithm A hands over the instance to Algorithm B and we constrain that $\mu_B(I) \leq \mu(I)$.

Compound analysis. We can view Wahlström's compound analysis [29] as a repeated application of Lemma 1. For example, there is one subroutine A_3 for when the maximum degree of the graph is 3. The algorithm prefers then to branch on a degree-3 vertex with all neighbors of degree 3. After all such vertices have been exhausted, the algorithm calls a new subroutine $A_{8/3}$ that takes as input a graph with maximum degree 3 where no degree-3 vertex has only degree-3 neighbors. In this case the average degree of the graph is at most $8/3$ - all neighbors of s in $\Gamma(G)$ are a (2,2,2) vertex in G - and the algorithm prefers to branch on vertices of degree 3 that have two neighbors of degree 3, etc. The analysis constrains that the measure for the analysis of $A_{8/3}$ is at most the measure for A_3 for the instance that is handed by A_3 to $A_{8/3}$. In an optimal analysis, we expect the measure for such an instance to be equal in the analysis of A_3 and $A_{8/3}$, and Wahlström actually imposes equality at the *pivot point* $8/3$.

Separate, Measure and Conquer. In our case, the $A_{8/3}$ algorithm is based on a technique from [20] known as *Separate, Measure and Conquer*. For small-degree graphs, we can compute small balanced separators in polynomial time. The algorithm then prefers to branch on vertices in the separator. The Separate, Measure and Conquer technique allows to distribute the large gain obtained by disconnecting the instance onto the previous branching vectors. While, often, the measure is made up of weights that are assigned to each vertex, this method assigns these weights only to the larger part of the graph that is separated from the rest by the separator, and somewhat larger weights to the vertices in the separator. See (3) on page 11 for the measure we use in the analysis. Thus, after exhausting the separator, the measure accurately reflects the "amount of work" left to do. We artificially increase the measure of very balanced instances by small penalty weights – this is done so because branching on vertices can change the measure of the parts that are separated by the separator and the branching strategy might not always be able to make most of its progress on the large side. Since we may exhaust the separators a logarithmic number of times, and computing a new separator might introduce a penalty term each time, the measure also includes a logarithmic term that counteracts these artificial increases in measure, and will in the end only contribute a polynomial factor to the running time. For an in-depth treatment of the method we refer to [20]. Since we use the *Separate, Measure and Conquer* method when the average degree drops to at most $8/3$, we slightly generalize the separation computation from [20], where the bound on the separator size depended only on the maximum degree. A separation (L, S, R) of a graph G is a partition of the vertex set into (possibly empty) sets L, S, R such that every path from a vertex in L to a vertex in R contains a vertex from S.

Lemma 2 *Let $B \in \mathbb{R}$. Let μ be a measure for graph problems such that for every graph $G = (V, E)$, every $R \subseteq V$, and every $v \in V$, we have that $|\mu(R \cup \{v\}) - \mu(R)| \leq B$. Assume that $\mu(R)$ can be computed in polynomial time. If there is an algorithm computing a path decomposition of width at most k of a graph G in polynomial time, where G contains a path decomposition of width k, then there is a polynomial time algorithm computing a separation (L, S, R) of G with $|S| \leq k$ and $|\mu(L) - \mu(R)| \leq B$.*

Proof The proof is the same as for the separation computation from [20], but we repeat it here for completeness. First, compute a path decomposition of width k in polynomial time. We view a path decomposition as a sequence of bags $(B_1, \ldots, B_b)$ which are subsets of vertices such that for each edge of G, there is a bag containing both endpoints, and for each vertex of G, the bags containing this vertex form a non-empty consecutive subsequence. The width of a path decomposition is the maximum bag size minus one. We may assume that $B_1 = B_b = \emptyset$ and that every two consecutive bags B_i, B_{i+1} differ by exactly one vertex, otherwise we insert between B_i and B_{i+1} a sequence of bags where the vertices from $B_i \setminus B_{i+1}$ are removed one by one followed by a sequence of bags where the vertices of $B_{i+1} \setminus B_i$ are added one by one; this is the standard way to transform a path decomposition into a *nice* path decomposition of the same width where the number of bags is polynomial in the number of vertices [5]. Note that each bag is a separator and a bag B_i defines the separation (L_i, B_i, R_i) with $L_i = (\bigcup_{j=1}^{i-1} B_j) \setminus B_i$ and $R_i = V \setminus (L_i \cup B_i)$. Since the first of these separations has $L_1 = \emptyset$ and the last one has $R_b = \emptyset$, at least one of these separations has $|\mu_r(L_i) - \mu_r(R_i)| \leq B$. Finding such a bag can clearly be done in polynomial time. $\square$

We will use the lemma for graphs with maximum degree 3 and graphs with maximum degree 3 and average degree at most $8/3$, for which path decompositions of width at most $n/6 + o(n)$ and $n/9 + o(n)$ can be computed in polynomial time, respectively [12, 14].

One disadvantage of using the Separate, Measure and Conquer method for $A_{8/3}$ is that the algorithm needs to choose vertices for branching so that the size of the separator decreases in each branch. However, Wahlström's algorithm defers to branch on degree-3 vertices with all neighbors of degree 2 until this is no longer possible, since this case leads to the largest branching factor for degree 3. For our approach, we instead rearrange the separator in some cases until we are only left with spider vertices, a structure where our algorithm cannot avoid branching on a degree-3 vertex with all neighbors of degree 2, we give a special weight to these spider vertices and upper bound their number.

Potentials. To optimize the running time further, we also use potentials; see [19, 22, 28]. These are constant weights that are added to the measure if certain global properties of the instance hold. For instance, we may use them to slightly increase the measure when an unfavorable branching strategy needs to be used. The constraint (2) for this unfavorable case then becomes less constraining, while all branchings that can lead to this unfavorable case get tighter constraints. This allows then to amortize unfavorable cases with favorable ones.

3 Algorithm

We first introduce notation necessary to present the algorithm. Let $V(G)$ and $E(G)$ denote the vertex set and the edge set of the input graph G. For a vertex $v \in V(G)$, its neighborhood, $N_G(v)$, is the set of vertices adjacent to v. The *closed neighborhood* of a vertex v is $N_G[v] = N_G(v) \cup \{v\}$. If G is clear from context, we use $N(v)$ and $N[v]$.

The degree of v is denoted $d(v) = |N_G(v)|$. For two vertices u and v connected by a path $P \subseteq V(G)$, if $P \setminus \{u, v\}$ consists only of degree-2 vertices then we call P a *2-path* of u and v.

The maximum degree of G is denoted $\Delta(G)$ and $d(G) = 2|E(G)|/|V(G)|$ is its *average degree*. A *cubic* graph consists only of degree-3 vertices. A *subcubic* graph has maximum degree at most 3. A $(k_1, k_2, ..., k_d)$ *vertex* is a degree-d vertex with neighbors of degree $k_1, k_2, ..., k_d$. A separation (L, S, R) of G is a partition of its vertex set into the three sets L, S, R such that no vertex in L is adjacent to any vertex in R. The sets L, S, R are also known as the *left set*, *separator*, and *right set*. Using a notion from [20], a separation (L, S, R) of G is *balanced* with respect to some measure μ, and a branching constant B if $|\mu(R) - \mu(L)| \leq 2B$ and *imbalanced* if $|\mu(R) - \mu(L)| > 2B$.

By convention, $\mu(R) \geq \mu(L)$ otherwise, we swap L and R. We will now describe the algorithm `#IS` which takes as input a graph G, a separation (L, S, R), and a cardinality function $\mathbf{c} : \{0, 1\} \times V(G) \to \mathbb{N}$, and computes the number of independent sets of G weighted by the cardinality function $\mathbf{c}$. For clarity, let $\mathbf{c}_{out}(v) = \mathbf{c}(0, v)$ and $\mathbf{c}_{in}(v) = \mathbf{c}(1, v)$. More precisely, it computes

$$ind(G, \mathbf{c}) = \sum_{\substack{X \subseteq V(G) \\ X \text{ is an independent set in } G}} \prod_{v \in X} \mathbf{c}_{in}(v) \cdot \prod_{v \in V \setminus X} \mathbf{c}_{out}(v).$$

Note that for a cardinality function $\mathbf{c}$ initialized to $\mathbf{c}(0, v) = \mathbf{c}(1, v) = 1$ for every vertex $v \in V(G)$, we have that $ind(G, \mathbf{c})$ is the number of independent sets of G. Cardinality functions are dynamic and used for bookkeeping during the branching process and have been used in this line of work before.

The separation (L, S, R) is initialized to $(\emptyset, \emptyset, V(G))$ and will only come into play when G is subcubic and has no (3,3,3)-vertex. In this case, the algorithm calls a subroutine `#3IS`, which constitutes the main contribution of this paper. `#3IS` computes a balanced separation of G, preferring to branch on vertices in the separator, readjusting the separator as needed, and is analyzed using the Separate, Measure and Conquer method.

Dragging refers to moving vertices or a set of vertices of G from one component of (L, S, R) to another, creating a new separation (L', S', R').

Skeleton Graph. The skeleton graph $\Gamma(G)$, or just Γ, of a subcubic graph G is a graph where the degree-3 vertices of G are in bijection with the vertices in Γ. Two vertices in Γ are adjacent if the corresponding vertices are adjacent in G, or there exists a 2-path between the corresponding vertices in G. If G has a separation (L, S, R) then denote $L_\Gamma, S_\Gamma, R_\Gamma$ to be the vertices of L, S, R respectively in Γ, which have exactly degree 3.

Spider Vertices. As Wahlström's [29] analysis showed, an unfavorable branching case occurs on (2, 2, 2) vertices. In this analysis we identified further specifically what kinds of (2, 2, 2) vertices were more undesirable, and attempted to amortize the weights of these vertices. We call these vertices *spider vertices*. Here Γ refers to the skeleton graph. A vertex s is a *spider vertex* if

- $s \in S$
- all neighbors of s are a (2,2,2) vertex and,
- either:
    - $|N_\Gamma(s) \cap L| = 2$ and $N_\Gamma(s) \cap R = \{r\}$ with r having neighbors of degree (2,2,2); in this case we call s a *left spider vertex*

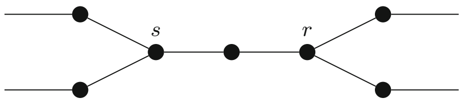

Fig. 1 A left spider vertex s

- $|N_\Gamma(s) \cap R| = 2$ and $N_\Gamma(s) \cap L = \{l\}$ with l having neighbors of degree (2,2,2); in this case we call s a *right spider vertex*
- $|N_\Gamma(s) \cap L| = 1$, $|N_\Gamma(s) \cap R| = 1$, $N_\Gamma(s) \cap S = \{s'\}$ and s' has neighbors of degree (2,2,2); in this case we call both s and s' a *center spider vertex*, which occur in pairs.

A left spider vertex $s \in S$ can be dragged to the left along with the 2-path from s to r. If this occurs, then r becomes a right spider vertex, and vice versa (Fig. 1).

Multiplier Reduction. We use a reduction called multiplier reduction to simplify graphs that have a cut vertex efficiently. Suppose $G = (V, E)$ has a separation $(V_1, \{x\}, V_2)$ and $G_1 = G[V_1 \cup \{x\}]$ has measure at most a constant B. For any vertex set $U \subseteq V$ and a subgraph $G' = (V', E')$ of G let $G'[U]$ represent $V' \cap U$, in other words, the vertices of U which remain in the new subgraph G'. The *multiplier reduction* can be applied to compute $\#\texttt{IS}(G, (L, S, R), \mathbf{c})$ as follows.

1. Let:
   - $G_{\text{out}} = G_1 \setminus \{x\}$
   - $G_{\text{in}} = G_1 \setminus N_{G_1}[x]$
   - $c_{\text{out}} = \#\texttt{IS}(G_{\text{out}}, (G_{\text{out}}[L], G_{\text{out}}[S], G_{\text{out}}[R]), \mathbf{c})$
   - $c_{\text{in}} = \#\texttt{IS}(G_{\text{in}}, (G_{\text{in}}[L], G_{\text{in}}[S], G_{\text{in}}[R]), \mathbf{c})$
2. Modify $\mathbf{c}$ such that $\mathbf{c}_{\text{in}}(x) = \mathbf{c}_{\text{in}}(x) \cdot c_{\text{in}}$ and $\mathbf{c}_{\text{out}}(x) = \mathbf{c}_{\text{out}}(x) \cdot c_{\text{out}}$
3. Return $\#\texttt{IS}(G[V_2 \cup \{x\}], (L \setminus V_1, S \setminus V_1, R \setminus V_1), \mathbf{c})$

Our measure will make sure that for any instance whose measure is upper bounded by a constant steps 1 and 2 can be performed in polynomial time. Since the measure of G_1 is upper bounded by a constant, G_1 is processed in polynomial time by the multiplier reduction.

Lazy 2-separator. Suppose there is a vertex x initially chosen to branch on as well as two vertices $\{y, z\} \subset V(G)$ with $d(y) \geq 3$ and $d(z) \geq 3$ such that x belongs to an induced subgraph of G of constant measure separated from the rest of the graph by the separator $\{y, z\}$. We call this separator a *lazy 2-separator*, for a vertex x. Similar to Wahlström's elimination of separators of size 2 in [28], in line 15 of $\#\texttt{IS}$ (Algorithm 1) instead of branching on x, if there exists a lazy 2-separator $\{y, z\}$ for x we branch on y. A multiplier reduction will be performed on z in the recursive calls. Prioritizing *lazy 2-separators* allows to exclude some unfavorable cases when branching on x.

Associated Average Degree. Similar to [29], we define the *associated average degree* of a vertex $x \in V(G)$ as $\alpha(x)/\beta(x)$, in G with average degree $d(G)$ where

$$\alpha(x) = d(x) + |\{y \in N(x) : d(y) < d(G)\}| \ , \text{ and}$$
$$\beta(x) = 1 + \sum_{y \in N(x),\, d(y) < d(G)} 1/d(y) \ .$$

By selecting vertices with high associated average degree, our algorithm prioritizes branching on vertices with larger decreases in measure.

Branching. We now outline the branching routine used to recursively solve smaller instances of the problem. Suppose we have a graph G, a separation (L, S, R), and a cardinality function $\mathbf{c}$. For a vertex x we denote the following steps as *branching on* x.

1. Let:
   - $G_{\text{out}} = G \setminus \{x\}$
   - $G_{\text{in}} = G \setminus N[x]$
   - $c_{\text{out}} = \#\texttt{IS}(G_{\text{out}}, (L[G_{\text{out}}], S[G_{\text{out}}], R[G_{\text{out}}]), \mathbf{c})$
   - $c_{\text{in}} = \#\texttt{IS}(G_{\text{in}}, (L[G_{\text{in}}], S[G_{\text{in}}], R[G_{\text{in}}]), \mathbf{c})$
   - $c'_{\text{out}} = \mathbf{c}_{\text{out}}(x)$
   - $c'_{\text{in}} = \mathbf{c}_{\text{in}}(x) \cdot \prod_{v \in N(x)} \mathbf{c}_{\text{out}}(v)$
2. Return $c'_{\text{out}} \cdot c_{\text{out}} + c'_{\text{in}} \cdot c_{\text{in}}$

Branching Vectors. Constraints, of the type referred to in (2), are presented as branching vectors (δ_1, δ_2) which equates to the constraints $2^{-\delta_1} + 2^{-\delta_2} \leq 1$.

4 Running Time Analysis

This section describes the running time analysis for #IS and #3IS, conducted via compound measures [29] using Lemma 1. Compound measures are piecewise measures which apply a finer analysis to specific states during the execution of the algorithm. In our case, the piecewise nature of compound measures allows different analyses and running times to apply for different average degrees of the input graph.

4.1 Measures

Measure with no (3,3,3) vertex. When using the Separate, Measure and Conquer technique from [20] the measure of a cubic graph instance G with no (3,3,3) vertices consists of additive components μ_s and μ_r, the measure of vertices in the separator, and those in either L or R, respectively. Let $S' \subseteq S$ be the set of all spider vertices, s_i and r_i refer to the weight attributed to a separator vertex and a right vertex, in R or L, respectively, of degree i. Left and right spider vertices have weight s'_3. In a center spider vertex pair s and s', one of them has weight s'_3 while the other takes on an ordinary weight of s_3. The weights s_3 and s'_3 attributed to spider vertices allows for amortization of the spider vertex cases against non-spider vertices. Define the measure $\mu_{8/3}$ as

$$\mu_{8/3} = \mu_s(S) + \mu_r(R) + \mu_o(L, S, R), \text{ where}$$
$$\mu_s(S) = |S'| \cdot s'_3 + \sum_{v \in S \setminus S'} s_{d(v)},$$

```
Algorithm: #IS($G$, $(L, S, R)$, $\mathbf{c}$) - #INDEPENDENT SET algorithm
Input     : Graph $G = (V, E)$, separation $(L, S, R)$ of $G$, cardinality function $\mathbf{c}$
Output    : $ind(G, \mathbf{c})$
1  if $V = \emptyset$ then
2      return 1
3  if $|V| = 1$ then
4      return $\mathbf{c}_{\text{in}}(x) + \mathbf{c}_{\text{out}}(x)$ where $V = \{x\}$
5  if $\Delta(G) \le 2$ then
6      return a solution in polynomial time
7  else if G is not connected and has j connected components $G_1, G_2, \dots, G_j$ then
8      return $\prod_{i=1}^{j}$ #IS($G_i$, $(\emptyset, \emptyset, V(G_i))$, $\mathbf{c}$)
9  else if $\Delta(G) = 4$, and all degree-4 vertices of G only have degree-2 neighbors, and there exists a
   vertex x where $d(x) = 4$ and x has a 2-path to a degree-3 vertex then
10     Branch on $x$
11 else
12     Let vertex $x \in V$ be a vertex of maximum degree, secondarily maximizing the associated
       average degree $\alpha(x)/\beta(x)$
       if the multiplier reduction applies then
13         Apply the multiplier reduction
14     else if there exists a separator of size 2: $\{y, z\}$, with $d(y) \ge 3$ and $d(z) \ge 3$ whose removal
       leaves G disconnected and either removes or leaves $N_G[x]$ in a component with constant
       measure at most $B$ then
15         Branch on $y$.
16     else
17         if $\Delta(G) = 3$ and G has no (3,3,3) vertex then
18             return #3IS($G$, $(L, S, R)$, $\mathbf{c}$)
19         else
20             Branch on $x$
```

Algorithm 1: Algorithm for counting independent sets of a graph G

$$\mu_r(R) = \sum_{v \in R} r_{d(v)},$$

$$B = 6s_3, \text{ and}$$

$$\mu_o(L, S, R) = \max\left\{0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right\} + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)). \tag{3}$$

We also require that $s_i \ge s_{i-1}$ and $r_i \ge r_{i-1}$ for $i \in \{1, 2, 3\}$. The constant $B = 6s_3$ is larger than the maximum change in imbalance in each transformation in the analysis, except the separation transformation.

The constant $\epsilon > 0$ is required to satisfy

$$\mu_r(R) + \mu_r(S) \ge (1 + \epsilon)(\mu_r(R') + \mu_s(S'))$$

which constrains that separating $(\emptyset, S, R)$ to (L', S', R') should reduce $\mu_r(R) + \mu_s(S)$ by a constant factor, namely $1 + \epsilon$.

```
Algorithm: #3IS($G$,($L, S, R$), $\mathbf{c}$) - #INDEPENDENT SET algorithm for subcubic graphs with no
           (3,3,3) vertex
Input   : Graph $G = (V, E)$, separation $(L, S, R)$ of $G$, cardinality function $\mathbf{c}$
Output  : $ind(G, \mathbf{c})$
1  if $S = \emptyset$ then
2      Compute balanced separation $(L, S, R)$ w.r.t. the measure $\mu$ using Lemma 2.
3  if $\mu_r(L) > \mu_r(R)$ then
4      Swap $L$ and $R$
5  $(L, S, R) :=$ simplify$(G, (L, S, R))$
6  if the multiplier reduction applies then
7      Apply the multiplier reduction.
8  Let $s \in S$ be a maximum degree vertex with maximum associated average degree.
9  if there exists a separator of size 2: $\{y, z\}$, with $d(y) \geq 3$ and $d(z) \geq 3$ whose removal leaves G
   disconnected and either removes or leaves $N_G[s]$ in a component with constant measure at most B
   then
10     Branch on $y$.
11 else if $\mu_r(R) - \mu_r(L) \leq 2B$ and s has neighbors of degree (2,2,2) then
12     return spider$(s, G, (L, S, R), \mathbf{c})$
13 else if $\mu_r(R) - \mu_r(L) > 2B$ and s has two neighbors in L and one neighbor r in R, let $r'$ be the
   first degree-3 vertex or vertex from S encountered when moving from s to the right along a 2-path P
   in G then
14     return #IS$(G, (L \cup P \cup \{s\}, (S \setminus \{s\}) \cup \{r'\}, R \setminus (P \cup \{r'\})), \mathbf{c})$
15 else if $\mu_r(R) - \mu_r(L) > 2B$ and there exists $r \in N_\Gamma(s) \cap R$ with $N_\Gamma(r) \cap R = \emptyset$ then
16     Let $\{r, r'\} = N_\Gamma(s) \cap R$ with $N_\Gamma(r) \cap R = \emptyset$
17     Branch on $r'$
18 else
19     Branch on $s$
```

Algorithm 2: Subroutine of #IS for counting independent sets in graphs of maximum degree 3 which have no (3,3,3) vertex.

We use the measure μ_r defined on page 11 to compute the separation in our algorithm using Lemma 2.

Lemma 3 *For a balanced separation (L, S, R) of a graph G, computed by Lemma 2, with average degree $d = d(G)$, maximum degree at most 3, and no (3,3,3) vertex, an upper bound for the measure $\mu_{8/3}$ is:*

```
Algorithm: simplify($G, (L, S, R)$) - Applies simplification rules.
Input    : Graph $G = (V, E)$, separation $(L, S, R)$ of $G$
Output   : $(L, S, R)$
1  if there exists a vertex $s \in S$ with no neighbor in $L$ then
2      return simplify($L, S \setminus \{s\}, R \cup \{s\}$)
3  else if there exists a vertex $s \in S$ with no neighbor in $R$ then
4      return simplify($L \cup \{s\}, S \setminus \{s\}, R$)
5  else if there exists a vertex $s \in S$ with $d(s) = 2$ then
6      if $(L, S, R)$ is balanced then
7          Let $l \in N_\Gamma(s) \cap (L \cup S)$. Let $P$ be the 2-path for $s$ and $l$.
8          return simplify($G, (L \setminus (P \cup \{l\}), (S \setminus \{s\}) \cup \{l\}, R \cup P \cup \{s\})$)
9      else
10         Let $r \in (N_\Gamma(s) \cap R) \cup S$. Let $P$ be the 2-path connecting $s$ and $r$.
11         return simplify($G, (L \cup P \cup \{s\}, (S \setminus \{s\}) \cup \{r\}, R \setminus (P \cup \{r\}))$)
12 else if there exists a vertex $s \in S$ such that for every vertex $l \in N_\Gamma(s) \cap L$ we have that
   $N_\Gamma(l) \cap L = \emptyset$ then
13     For $l \in (N_\Gamma(s) \cap L)$, let $A_l = (N_\Gamma(l) \cap S)$, let $P_{(s,l)}$ be the 2-path from $s$ to $l$, and for $a \in A_l$ let
       $P_{(l,a)} \subset V(G)$ be the 2-path from $l$ to $a$.
14     Let $B = (N_\Gamma(s) \cap S)$ and for $b \in B$ let $Q_b \subset V(G)$ be the 2-path from $s$ to $b$.
15     Let $C = \left(\bigcup_{b \in B} Q_b\right) \cup (N_\Gamma(s) \cap L) \cup \left(\bigcup_{l \in N_\Gamma(s) \cap L} A_l \cup P_{(s,l)} \cup \left(\bigcup_{a \in A_l} P_{(l,a)}\right)\right)$
16     return simplify($G, (L \setminus C, S \setminus (\{s\} \cup C), R \cup \{s\} \cup C)$)
17 else if there exists a vertex $s \in S$ such that for every vertex $r \in N_\Gamma(s) \cap R$ we have that
   $N_\Gamma(r) \cap R = \emptyset$ then
18     For $r \in (N_\Gamma(s) \cap R)$, let $A_r = (N_\Gamma(r) \cap S)$, let $P_{(s,r)}$ be the 2-path from $s$ to $r$, and for $a \in A_r$
       let $P_{(r,a)} \subset V(G)$ be the 2-path from $r$ to $a$.
19     Let $B = (N_\Gamma(s) \cap S)$ and for $b \in B$ let $Q_b \subset V(G)$ be the 2-path from $s$ to $b$.
20     Let $C = \left(\bigcup_{b \in B} Q_b\right) \cup (N_\Gamma(s) \cap R) \cup \left(\bigcup_{r \in N_\Gamma(s) \cap R} A_r \cup P_{(s,l)} \cup \left(\bigcup_{a \in A_r} P_{(r,a)}\right)\right)$
21     return simplify($G, (L \cup \{s\} \cup C, S \setminus (\{s\} \cup C), R \setminus C)$)
22 else
23     return $(L, S, R)$
```

Algorithm 3: Subroutine for #3IS which applies simplification rules.

$$\mu_{8/3}(d) \leq \begin{cases} \frac{n}{6}(d-2)s_3' + \frac{1}{2}\left(\frac{5n}{6}(d-2)r_3 + n(3-d)r_2\right) & \\ +\mu_o(L,S,R) + o(n) & \text{if } 2 \leq d \leq \frac{28}{11} \\ \frac{n}{4}(8-3d)s_3' + \frac{n}{12}(11d-28)s_3 & \\ +\frac{1}{2}\left(\frac{5n}{6}(d-2)r_3 + n(3-d)r_2\right) + \mu_o(L,S,R) + o(n) & \text{if } \frac{28}{11} < d \leq \frac{8}{3} \end{cases}$$

which is maximised when $d = \frac{8}{3}$ *with the value*

$$\mu_{8/3} \leq \frac{n}{9}s_3 + \frac{1}{2}\left(\frac{5n}{9}r_3 + \frac{n}{3}r_2\right) + \mu_o(L,S,R) + o(n)$$

if constraints $\frac{r_2}{2} \leq \frac{s_3'}{11} + \frac{5r_3}{22} + \frac{5r_2}{22} \leq \frac{s_3}{9} + \frac{5r_3}{18} + \frac{r_2}{3}$ *are satisfied.*

```
Algorithm: spider($s$, $G$, $(L, S, R)$, $\mathbf{c}$) - handles the (2,2,2) vertex $s$
Input    : A (2,2,2) vertex $s$, graph $G = (V, E)$, separation $(L, S, R)$ of $G$, cardinality function $\mathbf{c}$
Output   : $ind(G, \mathbf{c})$
if $|N_G(s) \cap R| = 1$ then
    Let $\{r\} = N_\Gamma(s) \cap R$ and let $P_r$ be the 2-path from $s$ to $r$.
    if $r$ is a (2,2,3) vertex or a (2,3,3) vertex then
        return #3IS($G$, $(L \cup P_r \cup \{s\}, (S \cup \{r\}) \setminus \{s\}, R \setminus (P_r \cup \{r\}))$, $\mathbf{c}$)
else if $|N_G(s) \cap L| = 1$ then
    Let $\{l\} = N_\Gamma(s) \cap L$ and let $P_l$ be the 2-path from $s$ to $l$.
    if $l$ is a (2,2,3) vertex or a (2,3,3) vertex then
        return #3IS($G$, $(L \setminus (P_l \cup \{l\}), (S \cup \{l\}) \setminus \{s\}, R \cup P_l \cup \{s\})$, $\mathbf{c}$)
else if $|N_\Gamma(s) \cap S_\Gamma| = 1$ then
    Due to simplify call, let $\{s'\} = N_\Gamma(s) \cap S_\Gamma$, $\{l\} = N_\Gamma(s) \cap L_\Gamma$ and $\{s, s_1, s_2\} = N_\Gamma(l)$.
    for $i \in \{1, 2\}$ do
        if $s_i \in S$ and $|N_G(s_i) \cap R| = 1$ then
            Let $\{r_i\} = N_G(s_i) \cap R$
            $(L, S, R) := (L \cup \{s_i\}, (S \cup \{r_i\}) \setminus \{s_i\}, R \setminus \{r_i\})$
    Branch on $l$.
else
    Branch on $s$.
```

Algorithm 4: Subroutine of #3IS which handles spider vertices.

Proof Let $d = d(G)$ be the average degree of G. For an appropriate upper bound of $\mu_{8/3}$ we first consider the upper bound on the number of separator vertices,

$$\#\text{Spiders} \le |S| \le \frac{n_3}{6} + o(n_3) = \frac{n(d-2)}{6} + o(n) \tag{4}$$

where $n_3 = n(d-2)$ is the number of degree-3 vertices in G, since a subcubic graph with n_3 vertices of degree 3 has pathwidth at most $\frac{n_3}{6} + o(n_3)$ [12].

As we have no (3,3,3) vertex, every degree-3 vertex is incident to an edge incident to a degree-2 vertex. However, each spider vertex has to have 4 more edges incident to degree-2 vertices. As the number of edges incident to degree-2 vertices is $2n_2$ where $n_2 = n(3-d)$ is the number of degree-2 vertices in G, and there are at least n_3 of those edges taken up to be incident to a degree-3 vertex, an upper bound on the number of spider vertices is:

$$\#\text{Spiders} \le \frac{2n_2 - n_3}{4} = n\left(2 - \frac{3}{4}d\right) \tag{5}$$

Since both upper bounds are valid for all $2 \le d \le 8/3$, a more accurate upper bound can be found by taking the minimum of Eqs. (4) and (5). This results in:

$$\#\text{Spiders} \le \begin{cases} \frac{n}{6}(d-2) + o(n) & \text{if } 2 \le d \le \frac{28}{11} \\ n\left(2 - \frac{3}{4}d\right) & \text{if } \frac{28}{11} < d \le \frac{8}{3} \end{cases}$$

As $|S| \leq \frac{n}{6}(d-2)+o(n)$ for all $2 \leq d \leq \frac{8}{3}$, with the weight for spider vertices s_3 being greater than regular non-spider degree-3 vertices in the separator, then an upper bound for $\mu_{8/3}$ would have as many spider vertices in S as possible for a given average degree d. For $2 \leq d \leq \frac{28}{11}$ it is possible to have all vertices in S be spider vertices, so this gives the greatest value of $\mu_{8/3}$. However, from $\frac{28}{11} < d \leq \frac{8}{3}$ we use Eq. (4) to upper bound $|S|$. We also place in S as many spider vertices with weight s_3' as Eq. (5) allows, with the rest of the vertices in S being of weight s_3. Therefore,

$$\mu_{8/3} \leq \begin{cases} \frac{n}{6}(d-2)s_3' + \frac{1}{2}\left(\frac{5n}{6}(d-2)r_3 + n(3-d)r_2\right) & \\ +\mu_o(L,S,R) + o(n) & \text{if } 2 \leq d \leq \frac{28}{11} \\ \frac{n}{4}(8-3d)s_3' + \frac{n}{12}(11d-28)s_3 + & \\ \frac{1}{2}\left(\frac{5n}{6}(d-2)r_3 + n(3-d)r_2\right) + \mu_o(L,S,R) + o(n) & \text{if } \frac{28}{11} < d \leq \frac{8}{3}. \end{cases}$$

Let $f_1(d) = \frac{n}{6}(d-2)s_3' + \frac{1}{2}(\frac{5n}{6}(d-2)r_3 + n(3-d)r_2)$ and $f_2(d) = \frac{n}{4}(8-3d)s_3' + \frac{n}{12}(11d-28)s_3 + \frac{1}{2}(\frac{5n}{6}(d-2)r_3 + n(3-d)r_2)$. We notice that f_1 and f_2 are both linear functions in d and $f_1(\frac{28}{11}) = f_2(\frac{28}{11})$ meaning that the endpoints $f_1(2)$, $f_2\left(\frac{28}{11}\right)$, and $f_2\left(\frac{8}{3}\right)$ are the only points of interest. For the piecewise measure to be valid, and not result in an increase of measure as d decreases, we consider the values of $\mu_{8/3}$ at the endpoint values of 2, $\frac{28}{11}$, and $\frac{8}{3}$ and require that $f_1(2) \leq f_2\left(\frac{28}{11}\right) \leq f_2\left(\frac{8}{3}\right)$ which results in the constraints

$$\frac{r_2}{2} \leq \frac{s_3'}{11} + \frac{5r_3}{22} + \frac{5r_2}{22} \leq \frac{s_3}{9} + \frac{5r_3}{18} + \frac{r_2}{3}.$$

Also the maximum value achieved by f_2 when average degree $d = \frac{8}{3}$ is:

$$\begin{aligned} \mu_{8/3} &\leq f_2\left(\frac{8}{3}\right) + \mu_o(L,S,R) + o(n) \\ &= \frac{n}{9}s_3 + \frac{1}{2}\left(\frac{5n}{9}r_3 + \frac{n}{3}r_2\right) + \mu_o(L,S,R) + o(n). \end{aligned}$$

□

General Measure In order to analyze higher degree cases, we use a measure of the form

$$\mu_i(G) = \sum_{v \in G} r_{d(v)} + \mu_o(L,S,R) \quad \text{where } \Delta(G) = i$$

for each part of the compound measure, as defined by the parameter i, which represents different branching cases. When branching, we always attempt to find the highest degree vertex, with neighbors of highest degrees. Once this kind of vertex is exhausted, we will never branch on this kind of vertex again, and transition to a different case and apply a different measure $\mu_i(G)$. These cases, described in Table 1, are defined by their highest possible average degree.

The term $\mu_o(L, S, R)$ is the same sub-linear term from the Separate, Measure and Conquer analysis in Eq. (3) which needs to be propagated into the higher degree analyses.

4.2 Degree 3 Analysis

The problem of counting independent sets, #IS, can be solved in polynomial time when $\Delta(G) \leq 2$ [26]. However, stepping up to cubic graphs is a much harder problem. Greenhill [21] proves that #3IS is actually a #P-hard problem.

Lemma 4 *Algorithm* `#IS` *applied to a graph G with* $\Delta(G) \leq 3$ *and no* $(3, 3, 3)$ *vertex has running time* $O(1.0963^n)$.

Proof Lemma 4 will be proved over the next few subsections. We will analyze the running time with respect to the measure $\mu_{8/3}$ described in Eq. (3). Weights will be attributed to vertices, depending on structural properties such as their degree, and whether or not they are *spider vertices*. As suggested in [20] we will provide constraints that these vertex weights need to satisfy, and the provided values minimize an upper bound of the measure of the form αn. The measure $\mu_{8/3}$ can be viewed in two regimes; a balanced separation, where $\mu_r(R) - \mu_r(L) \leq 2B$ resulting in $\mu_{8/3} = \mu_s(S) + \frac{1}{2}(\mu_r(R) - \mu_r(L)) + \mu_o(L, S, R)$ and an imbalanced separation, where $\mu_r(R) - \mu_r(L) > 2B$ resulting in $\mu_{8/3} = \mu_s(S) + \mu_r(R) + \mu_o(L, S, R)$. To characterize decreases in vertex degrees, let $\Delta s_i = s_i - s_{i-1}$ and $\Delta r_i = r_i - r_{i-1}$. Trivial constraints are

$$r_0 = r_1 = 0 \qquad s_0 = s_1 = 0.$$

Our algorithm handles 2-paths as if they were single edges. This is due to the consideration of the skeleton graph which is used when making decision on how to branching. By using the skeleton graph in the algorithm, which imagines 2-paths as single edges, we also constrain that $r_2 = 0$. This means that degree 2 vertices do not impact the measure since we won't need to branch on these kinds of vertices and only ever simplify them.

Constraints from `#IS` Simplification rules in lines 2 to 8 in `#IS` take polynomial time. If we are given a graph G with $\Delta(G) \leq 3$ and no $(3, 3, 3)$ vertex and the lazy 2-separator rule in line 15 did not apply, then we enter the subroutine `#3IS`.

Constraints from `simplify` The simplification rules in `simplify` either reduce the separator size by removing a vertex or the rule drags degree-2 vertices in S away making S consist only of degree-3 vertices. For vertex dragging to R in line 2 of `simplify`, the most constraining instances are the balanced ones:

$$-s_d + r_d \leq 0 \text{ where } d \in \{2, 3\} \text{ and } -s_3' + r_3 \leq 0\ .$$

However, for vertex dragging to L in line 4, the imbalanced instances are most constraining

$$-s_d + 1/2 \cdot r_d \leq 0 \text{ where } d \in \{2, 3\} \text{ and } -s'_3 + 1/2 \cdot r_3 \leq 0$$

but this is no more constraining than the constraints generated from line 2 of `simplify`.

Line 8 drags to R the degree-2 separator vertex s and a 2-path, ending in a vertex l which is either in S or has degree 3, which itself is dragged into S. This most constraining in the balanced case

$$-s_2 + s'_3 + \frac{1}{2} \cdot (r_2 - r_3) \leq 0;$$

In line 11 the most constraining case is

$$-s_2 + s'_3 - r_3 \leq 0 \ .$$

The operations in line 16 drag neighbors and associated 2-paths from L into R, also removing $s \in S$. Since $r_2 = 0$ we can simplify the most constraining case, which is imbalanced, to:

$$-s_3 + 2r_3 \leq 0.$$

Line 21 is most constraining in the balanced case, which induces the constraint

$$-s_3 \leq 0.$$

Claim *After* `simplify` *has been applied to a graph G and its separation (L, S, R), for each $s \in S$ there exists $r \in N_\Gamma(s) \cap R$ such that $N_\Gamma(r) \cap R \neq \emptyset$, and also there exists $l \in N_\Gamma(s) \cap L$ such that $N_\Gamma(l) \cap L \neq \emptyset$*

Proof If there is a vertex s that does not satisfy the claim, then line 16 or 21 would trigger and remove s from S. □

Constraints from `spider` The first two conditions of lines 3 and 7 in `spider` aim to drag into the separator a (2,2,3) or (2,3,3) vertex in order to branch more efficiently on. In the worst case there is no change in measure since s is replaced by r in the separator. Since the separation (L, S, R) is balanced, due to the context in which `spider` is called, moving P_r and r or P_l and l also does not change the measure as L and R contribute equally to $\mu_{8/3}$.

In line 14, s is a center spider vertex with attributed weight s'_3. We branch on $l \in L$, which is a skeleton neighbor of s. The for loop drags vertices which are skeleton neighbors of l with no change in measure so that when l is branched on, it obtains a decrease in measure of at least $\frac{3}{2}r_3$ by its neighbors. However, we choose l to branch on because on both subproblems, branching on l causes the removal of s from the

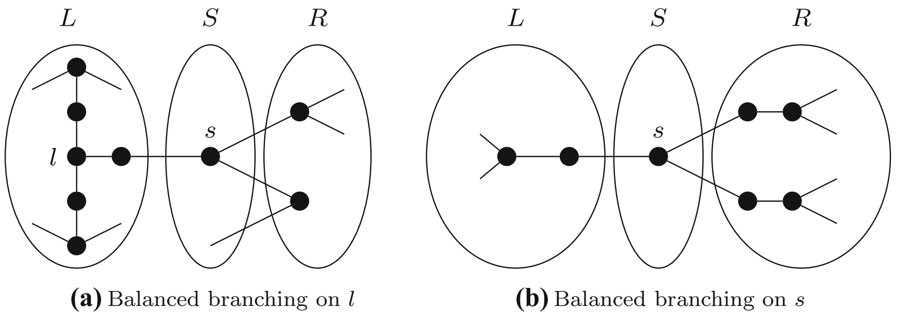

Fig. 2 Worst case configurations for spider vertex branching in `spider`

separator as it no longer has neighbors in L. This results in the branching constraints, described in Fig 2 (recall from Page 9):

$$\left(s_3' + \frac{1}{2}(r_3 + 2\Delta r_3), s_3' + \frac{1}{2}(r_3 + 2\Delta r_3)\right) .$$

Line 17 finds a valid left or right spider vertex and branches on it, resulting in the constraints

$$\left(s_3' + \frac{3}{2}\Delta r_3, s_3' + \frac{3}{2}\Delta r_3\right) .$$

Constraints from `#3IS` *- Computing Separator.* Much like in [20], computing a new separator in line 2 of `#3IS` imposes the constraint

$$s_3'/6 + 5/12 \cdot r_3 < r_3, \text{ or } \quad s_3' < 7/2 \cdot r_3.$$

In line 5 the algorithm simplifies the graph G and its separation (L, S, R) through a call to `simplify` (Algorithm 3). This algorithm applies simplification rules which also imposes new constraints to be satisfied for the analysis.

The constraints for the reduction rule in line 14 are the same as the constraints for line 11 in `simplify`. We now deal with branching on lazy-2 separators and regular branching, in both imbalanced and balanced cases, separately. As decreasing a degree-3 vertex weight to a degree-2 vertex weight may result in the introduction of a spider vertex with weight s_3' instead of s_3, let $\delta = s_3' - s_3$ be the increase in measure from a spider vertex creation. This increase in measure via δ is offset by either a Δs_3 or $\frac{1}{2}\Delta r_3$ decrease in measure in the same constraint.

Constraints from `#3IS` *- Balanced Lazy 2-Separator Branching* Suppose the instance is balanced and `#3IS` selects a vertex $s \in S$ but s has a lazy 2-separator $\{y, z\}$ which line 10 of `#3IS` branches on instead of s. As the degree-3 vertices y, z and s are all removed in the branches of this problem, as well as the fact that due to Claim 4.2 for L and R there will be another degree-3 vertex that will be removed (see described in

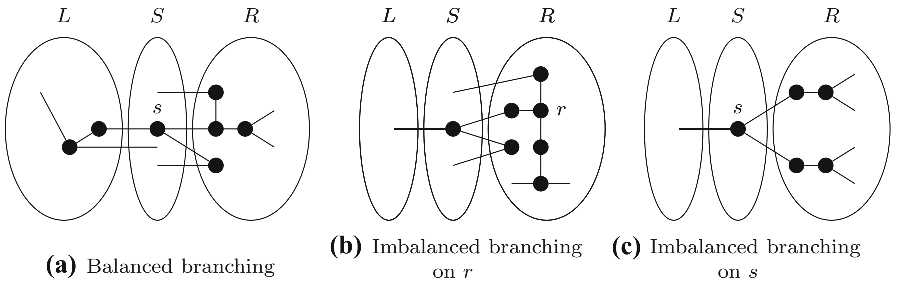

Fig. 3 Worst case configurations for non-spider vertex branching in `#3IS`

Fig 3) we obtain the branching vector

$$\left(s_3 + \frac{1}{2}(2r_3 + 2\Delta r_3) - 2\delta, s_3 + \frac{1}{2}(2r_3 + 2\Delta r_3) - 2\delta\right).$$

The worst case contains measure increases of 2δ since the two decreases of $\frac{1}{2}\Delta r_3$ could create a spider vertex, and there are at least 2 of them. We could have more δ decreases, but this only occurs when we have an additional $\frac{1}{2}\Delta r_3$ decrease, or Δs_3 decrease in the worst case. But since $\delta \leq \Delta s_3 \leq \frac{1}{2}\Delta r_3$ the tightest constraint occurs at the smallest possible number of δ changes.

Constraints from #3IS - Imbalanced Lazy 2-Separator Branching Once again, we have a vertex $s \in S$ and a lazy 2-separator $\{y, z\}$, but the instance is imbalanced. First assume either 1 or more of $\{y, z\}$ is in R. In this case, we disconnect s, either the y or z vertex y, as well as some other vertex $r \in R$ due to Claim 4.2. At worst this results in the branching vector

$$(s_3 + 2r_3, s_3 + 2r_3).$$

In the case where $\{y, z\} \subseteq L$ we also refer to Claim 1 which guarantees that there is a skeleton neighbor $r \in N_\Gamma(s) \cap R$, which itself has a neighbor $r' \in N_\Gamma(r) \cap R$. These two combined with s are removed in both branches, otherwise s cannot be removed and $\{y, z\}$ is not a lazy-2 separator. This also results in the branching vector

$$(s_3 + 2r_3, s_3 + 2r_3).$$

Constraints from #3IS - Balanced Branching: neighbor in separator Consider the balanced branching case where we branch on $s \in S$ and s has a neighbor $s' \in S$. Let $u \in R$ and $v \in L$ denote the two other neighbors. In the worst case, u and v are both degree-2 vertices, meaning in both branches we only reduce a vertex of weight r_3 to r_2, but never delete one. Since s' reduces in degree in the first branch and is removed in the second branch, then we get the following branching vector

$$\left(s_3 + \Delta s_3 + \frac{1}{2}(2\Delta r_3) - 3\delta, 2s_3 + \frac{1}{2}(2\Delta r_3) - 2\delta\right).$$

4.2.1 Constraints from #3IS - Balanced Branching: no Neighbor in Separator

Next consider the balanced branching case where the algorithm branches on a non-spider vertex $s \in S$ with no neighbors in the separator S. Let $u, u' \in R$ and $v \in L$ denote its neighbors. Since s is not a spider vertex, s is either a (2,2,3) or a (2,3,3) vertex.

We first consider the case where s is a (2,2,3) vertex. In the worst case, the single degree-3 neighbor of weight $\frac{r_3}{2}$ would be in R or L. This leads to a decrease of $\frac{\Delta r_3}{2}$. Of the two remaining neighbors, they are the start of a 2-path to another degree-3 vertex. Now both of these cannot be in S so we will have another decrease of at least $\frac{\Delta r_3}{2}$, leaving a decrease of Δs_3 for the last neighbor.

In the second case, we also get a decrease of $\Delta s_3 + \frac{\Delta r_3}{2}$ from the degree-3 neighbor of s. This is due to Claim 4.2 forcing at least 1 of the neighbors to be in R. This results in a branching vector of

$$\left(s_3 + \Delta s_3 + \frac{1}{2}(2\Delta r_3) - 3\delta, s_3 + 2\Delta s_3 + \frac{1}{2}(r_3 + 2\Delta r_3) - 4\delta\right).$$

Now if s is a (2,3,3) vertex, s has 2 degree 3. In the worst case, the degree 2 neighbor of s is the start of a 2-path to another vertex in S.

$$\left(s_3 + \Delta s_3 + \frac{1}{2}(2\Delta r_3) - 3\delta, s_3 + 3\Delta s_3 + \frac{1}{2}(2r_3 + 2\Delta r_3) - 5\delta\right).$$

Constraints from #3IS - Imbalanced Branching: neighbor in separator In the imbalanced instances of G the measure $\mu_{8/3}$ simplifies to $\mu_{8/3} = \mu_s(S) + \mu_r(R) + \mu_o(L, S, R)$. Suppose we choose $s \in S$ to branch on and s has a neighbor $s' \in S$. By Claim 4.2, s has a skeleton neighbor $r \in N_\Gamma(s) \cap R$. Now in the worst case, r is only a skeleton neighbor, and the actual neighbor $r' \in N_G(s) \cap R$ is of degree 2. By considering the removal, or reduction of degree, of s, s' and r', then we get the following worst case constraint

$$(s_3 + \Delta s_3 + r_3 - 3\delta, 2s_3 + r_3 + 5\delta).$$

The first branch has a 3δ term since we get at most 1 decrease for each neighbor. The 5δ term comes from the fact that the left neighbor $l \in N_G(s) \cap L$ does not contribute any weight to $\mu_{8/3}$ meaning it could have degree 3. Now s' is also of degree 3, so in the second case where we remove s' and l, these two could create 4 spider vertices. The last possible increase comes from r being reduced to a degree-2 vertex.

Constraints from #3IS - Imbalanced Branching: no neighbors in separator There are two branching rules to consider in this case. First branching occurs in line 17 where instead of branching on $s \in S$ we branch on one of its skeleton neighbors in R. The other case occurs when we branch on s as normal in line 19.

In line 17, we are given the case where s has 1 skeleton neighbor in R. This means that we do not get a beneficial branching by branching on s. However, in a similar method to line 15 of `spider`, if we branch on $r \in N_\Gamma(s) \cap R$ such that $N_\Gamma(r) \cap R \neq \emptyset$, then in both branches, we are able to remove s entirely from the separator due to the simplification rules in `simplify`. We get the following worst case constraint

$$(r_3 + s_3 + \Delta r_3 + \Delta s_3 - 3\delta, r_3 + s_3 + \Delta r_3 + \Delta s_3 - 3\delta).$$

Otherwise, we progress to line 19, which guarantees that we have 2 skeleton neighbors of s in R. This results in the following constraint

$$(s_3 + 2\Delta r_3 - 3\delta, s_3 + 2\Delta r_3 - 4\delta).$$

Weights and Results. After compiling all necessary constraints induced by the steps of the algorithms, we are able to attempt to set up a constraint optimization problem in order to minimize the measure, with respect to these constraints. The combination of all constraints obtained in this way, minimizing the measure results in the measure of $\mu_{8/3} = 0.13262 \cdot n$, and that the running time is $O(2^{\mu_{8/3}}) \subseteq O(2^{0.13262n})$ results in an upper bound of $O(1.0963^n)$ for the running time. The specific weights after minimization are summarized below.

$$\begin{array}{lllll} r_0 = 0 & r_1 = 0 & r_2 = 0 & r_3 = 0.2 + o(n) & \\ s_0 = 0 & s_1 = 0 & s_2 = 0.6352 & s_3 = 0.6784 & s'_3 = 0.7 \end{array}$$

□

Lemma 5 *Algorithm #*`IS` *applied to a graph G with $d(G) \leq 3$ has running time $O(1.1389^n)$ and uses polynomial space.*

The algorithm `#IS` uses subroutine `#3IS`, which we analyze the measure and the weights for. We equate the Separate, Measure and Conquer weights with weights of the measure μ_3, based on the compound analysis from Wahlström [29]. As Wahlström's analysis only contains weights w'_3 and w'_2, for vertices of degree 3 and degree 2 respectively, the measure is

$$\mu_3(G) = \big((d-2)w'_3 + (3-d)w'_2\big)\, n + \mu_o(L, S, R)$$

where $d = d(G)$ is the average degree of the input graph, and $\mu_o(L, S, R)$ is the sub-linear term left over from the average degree 8/3 analysis.

In the case of a graph G with no (3,3,3) vertex, in order for Lemma 1 to apply, the values of w_1 and w_2 must satisfy inequalities

$$\frac{1}{2} r_2 \leq w_2,$$
$$\frac{s'_3}{11} + \frac{5r_3}{22} + \frac{5r_2}{22} \leq \frac{6w_3}{11} + \frac{5w_2}{11},$$
$$\frac{s_3}{9} + \frac{5r_3}{18} + \frac{r_2}{3} \leq \frac{2w_3}{3} + \frac{w_2}{3},$$

induced when $d = 2, \frac{28}{11}$, and $\frac{8}{3}$ for $\mu_{8/3}$ respectively. This results in the weights $w_3 = 0.1973$ and $w_2 = 0.0033$ when G has no (3,3,3) vertex.

We also let $w'_3 \geq 0$ and $w'_2 \geq 0$ be the weights associated with vertices of degree 3 and degree 2 respectively, for a subcubic graph G. Using the analysis by compound measures with $\mu_3(G) = \sum_{i \in \{2,3\}} w'_i \cdot n_i$, the following constraint

$$\mu_{8/3}(G) \leq \mu_3(G), \quad \text{when } d(G) = 8/3$$

is required for a valid compound measure. This can be rewritten as

$$2w_3 + w_2 \leq 2w'_3 + w'_2.$$

Branching on a (3,3,3) vertex, the only type of degree-3 vertex that we will be branched on in `#IS`, gives a branching vector of

$$(4w'_3 - 3w'_2, 8w'_3 - 4w'_2).$$

Setting the weights $w'_3 = 0.1876$ and $w'_2 = 0.0228$ satisfies the system of constraints described above and by using the measure $\mu_3(G)$, results in a running time of $O^*(1.1389^n)$.

4.3 Degree-4 Analysis

The notation $\mathbb{1}(\cdot)$ refers to an indicator function which returns 1 if its argument is true and 0 otherwise. For a graph with maximum degree 4, the analysis is done with a measure of

$$\mu_4 = \sum_{i \leq 4} w_i \cdot n_i + \mathbb{1}\left(\begin{array}{l} G \text{ has only degree-4 and degree-2} \\ \text{vertices and no degree-4 vertex has} \\ \text{a degree-4 neighbor} \end{array}\right) \psi + \mu_o(L, S, R)$$

where w_i are weights attributed to vertices of degree i, n_i are the number of vertices with degree i and ψ is a potential.

We can ignore weights applied to degree 0 or degree 1 vertices since they are removed by simplification rules, and effectively have a weight of 0.

Potentials in Degree-4 Analysis. For analyzing branching on vertices in the degree-4 case, potentials are used for branching on a degree-4 vertices who only have degree-2 neighbors. In case (a), for any vertex v we have that all 2-paths starting from v, have endpoints of degree 4. Case (b) is where there exists one vertex u who has at least one 2-path from u that ends up in a degree-3 vertex. We do not have the case where a 2-path from a degree-4 vertex ends up in a degree-1 vertex since we could have used multiplier reduction to handle this case.

Lemma 6 *For a graph G with maximum degree 4, #IS can be solved in time* $O^*(1.2070^n)$.

Table 1 Possible cases when branching on a degree-4 vertex

Degrees of Neighbors	Highest Average Degree	Branching Vector
(2,2,2,2) (a)	3	$(5w_4 - 4w_3 + 4w_2 - \psi, 5w_4 - 4w_3 + 4w_2 - \psi)$
(2,2,2,2) (b)	3	$(4w_4 - 2w_3 + 3w_2 + \psi, 4w_4 - 2w_3 + 3w_2 + \psi)$
(2,2,2,3)	3	$(4w_4 - 2w_3 + 2w_1, 4w_4 - 2w_3 + 3w_2)$
(2,2,2,4)	3	$(5w_4 - 4w_3 + 3w_2, 6w_4 - 4w_3 + 3w_2)$
(2,2,3,3)	3	$(3w_4, 5w_4 - 2w_3 + 2w_2)$
(2,2,3,4)	3	$(4w_4 - 2w_3 + w_2, 5w_4 - 2w_3 + 2w_2)$
(2,2,4,4)	3	$(5w_4 - 4w_3 + 2w_2, 7w_4 - 4w_3 + 2w_2)$
(2,3,3,3)	16/5 = 3.2	$(2w_4 + 2w_3 - 2w_2, 4w_4 + w_2)$
(2,3,3,4)	42/13 ≈ 3.23	$(3w_4 - w_2, 6w_4 - 2w_3 + w_2)$
(2,3,4,4)	36/11 ≈ 3.27	$(4w_4 - 2w_3, 6w_4 - 2w_3 + w_2)$
(2,4,4,4)	10/3 ≈ 3.33	$(5w_4 - 4w_3 + w_2, 8w_4 - 4w_3 + w_2)$
(3,3,3,3)	24/7 ≈ 3.43	$(w_4 + 4w_3 - 4w_2, 5w_4)$
(3,3,3,4)	7/2 = 3.5	$(2w_4 + 2w_3 - 3w_2, 5w_4)$
(3,3,4,4)	18/5 = 3.6	$(3w_4 - 2w_2, 7w_4 - 2w_3)$
(3,4,4,4)	15/4 = 3.75	$(4w_4 - 2w_3 - w_2, 7w_4 - 2w_3)$
(4,4,4,4)	4	$(5w_4 - 4w_3, 9w_4 - 4w_3)$

Average Degree	w_2	w_3	w_4	Time
2-3	0.0227913	0.1875202	0.3295266	$O(1.13880^n)$
3-3.2	0.0659881	0.1875202	0.2863298	$O(1.15451^n)$
3.2-3.5	0.0795475	0.1897802	0.2772902	$O(1.17571^n)$
3.5-3.75	0.0911988	0.1936639	0.2734064	$O(1.19207^n)$
3.75-4	0.1057321	0.1998925	0.2713302	$O(1.2070^n)$

Fig. 4 Component measures $\sum_i w_i \cdot n_i$ for maximum degree 4

Proof The degree-4 analysis uses pivot points 3, 3.2, 3.5, 3.75 and 4, shown as different rows of Figure 4. These pivot points refer to the highest possible average degree in those respective branching cases, which are defined by the highest possible degree of neighbors.

Pivot points generate multiple compound measures with weights and constraints for each. By including constraints generated from the table of branching factors in Figure 1, we gain satisfying weights for μ_4, shown in Figure 4. This results in a running time upper bound of $O(2^{\mu_4 n}) \subseteq O(2^{0.2713n}) \subseteq O(1.2070^n)$ in the worst case for degree-4 graphs. □

4.4 Degree-5+ Analysis

The following two theorems show for degree-5+ graphs the generalized procedure for constructing branching vectors for v and all its possible combinations of degrees of neighbors.

Lemma 7 *Suppose a graph G is 3-connected, with all simplification rules applied. Let $v \in V(G)$ be a vertex to be branched on in $\#\mathtt{IS}$ with $d(v) \in \{5, 6\}$. Let $out(v)$ represent the number of edges xy from $N(v)$ with $x \in N(v)$ and $y \notin N[v]$. Then*

$$out(v) = \begin{cases} 3 & \textit{If } d(v) = 5 \textit{ and } \sum_{u \in N(v)} d(u) = 0 \textit{ mod } 2 \textit{ or} \\ & \quad d(v) = 6 \textit{ and } \sum_{u \in N(v)} d(u) = 1 \textit{ mod } 2 \\ 4 & \textit{If } d(v) = 5 \textit{ and } \sum_{u \in N(v)} d(u) = 1 \textit{ mod } 2 \textit{ or} \\ & \quad d(v) = 6 \textit{ and } \sum_{u \in N(v)} d(u) = 0 \textit{ mod } 2 \\ 5 & \textit{If neighbors of v have degree (2, 2, 2, 2, 2) or (2, 2, 2, 2, 2, 3)} \\ 6 & \textit{If neighbors of v have degree (2, 2, 2, 2, 2, 2),} \end{cases}$$

otherwise if $out(v)$ $leq2$ the graph G is constant sized.

Proof Let $out(v)$ represent the number of outgoing edges xy from $N(v)$ with $x \in N(v)$ and $y \notin N[v]$. Suppose we have a 3-connected graph with all simplification rules applied. This means that the multiplier reduction does not apply, and there are no lazy 2-separators.

If $out(v) = 0$ then we have an instance of size d, which is 5 or 6, and the graph is completely connected. This instance cannot occur, and even if it did, we would be able to solve it in constant time. If $out(v) = 1$ we can apply the multiplier reduction, which is a contradiction. Similarly, if $out(v) = 2$ we have a lazy 2-separator which is also a contradiction. Hence $out(v) \geq 3$.

Suppose $d(v) = 5$ and v has neighbors with degrees (2, 2, 2, 2, 2). Any edge adjacent to two neighbors of v means G can be reduced by multiplier reduction by branching on v, so $out(v) = 5$. Similarly, if $d(v) = 6$ and v has neighbors (2, 2, 2, 2, 2, 2), then $out(v) = 6$.

Supppose $d(v) = 6$ and v has neighbors (2, 2, 2, 2, 2, 3). There are 7 edges adjacent to $N(v)$ but not v. Suppose $u \in N(v)$ and $d(u) = 3$. If $out(v) < 5$ then at least 3 of these 7 edges must connect two vertices in $N(v)$, but at most two of them are adjacent to u. Thus there exists one edge $\{a, b\}$ with $d(a) = d(b) = 2$. But this means the multiplier reduction can be applied, hence $out(v) = 5$.

Suppose $d(v) = 5$ and $\sum_{u \in N(v)} d(u) = 1 \mod 2$. We showed there are at least 3 outgoing edges from $N(v)$. There are also 5 edges incident to $N(v)$ and v which gives a total of at least 8 edges that are incident to $N(v)$. Since having adjacent neighbors does not change the fact that $\sum_{u \in N(v)} d(u)$ is odd, $out(v)$ must be even.

If $\sum_{u \in N(v)} d(u)$ is odd, then since any edge adjacent to two neighbors of v contributes a value of 2 to the sum, then $\sum_{u \in N(v)} d(u) = 1$ mod 2 implies $out(v) = 4$. A similar parity argument is used for $d(v) = 6$, except with the parity swapped. □

Lemma 8 *Let $deg_2(v)$ denote the number of degree-2 vertices in $N(v)$. Then v has a branching vector of*

$$\left(w_{d(v)} + \sum_{u \in N(v)} w_{d(u)} + out(v) \cdot \Delta w_{d(v)}, w_{d(v)} + \sum_{u \in N(v)} \Delta w_{d(u)} + deg_2(v) \cdot \Delta w_{d(v)} \right). \tag{6}$$

Average Degree	w_2	w_3	w_4	w_5	w_6	Time
4-6	0.1146078	0.2017931	0.2713406	0.2977566	0.3051140	$O(1.2356^n)$

Fig. 5 Weights and running time for $\mu_6(G)$

Proof The left hand side of the branching factor considers removing a vertex v and its neighbors. The right hand side considers removing just a vertex v. The reduction in measure on the graph G follows from reduction rules, the measure $\mu = \sum_{v \in F} w_{d(v)} + \mu_o(L, S, R)$ and the definition of $out(v)$ and $deg_2(v)$. □

Theorem 1 *#IS can be solved in time* $O^*(1.2356^n)$ *and polynomial space.*

Proof If $d(G) \geq 7$ we can perform a simple branching analysis in terms of n, considering if a selected vertex v is inside the independent set, or its neighbors are. In this situation, the branching number is at worst $(1, 8) < 1.2321$. So we only need to compute $\mu_6(G)$ with compound measures using Eq. (6), with $d(G) \leq 6$ in order to find the worst case running time for #IS (Fig. 5).

If we plug in a simple pathwidth-based subroutine [14] for graphs of maximum degree 3, we obtain the following exponential-space result.

Theorem 2 *#IS can be solved in time* $O^*(1.2330^n)$.

Acknowledgements We thank Magnus Wahlström for clarifying an issue of the case analysis in [29] and an anonymous reviewer for useful comments on an earlier version of the paper. Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048) and acknowledges support under the ARC's Discovery Projects funding scheme (DP150101134). Edward Jay Lee is a recipient of an Australian Government Research Training Program Scholarship and acknowledges their support.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

References

1. Angelsmark, O., Thapper, J.: Partitioning based algorithms for some colouring problems. In: Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2005), Lecture Notes in Computer Science, vol 3978, pp. 44–58. Springer (2005) https://doi.org/10.1007/11754602_4
2. Björklund, A., Husfeldt, T.: Inclusion–exclusion algorithms for counting set partitions. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 575–582. IEEE Computer Society (2006) https://doi.org/10.1109/FOCS.2006.41
3. Björklund, A., Husfeldt, T.: Exact algorithms for exact satisfiability and number of perfect matchings. Algorithmica **52**(2), 226–249 (2008). https://doi.org/10.1007/s00453-007-9149-8

4. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. Comput. **39**(2), 546–563 (2009)
5. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms **21**(2), 358–402 (1996)
6. Bodlaender, H.L., Kratsch, D.: An exact algorithm for graph coloring with polynomial memory. Tech. Rep. UU-CS-2006-015, Department of Information and Computing Sciences, Utrecht University (2006)
7. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. Oper. Res. Lett. **32**(6), 547–556 (2004). https://doi.org/10.1016/j.orl.2004.03.002
8. Christofides, N.: An algorithm for the chromatic number of a graph. Comput. J. **14**(1), 38–39 (1971). https://doi.org/10.1093/comjnl/14.1.38
9. Dahllöf, V., Jonsson, P., Wahlström, M.: Counting models for 2SAT and 3SAT formulae. Theoret. Comput. Sci. **332**(1–3), 265–291 (2005). https://doi.org/10.1016/j.tcs.2004.10.037
10. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. Journal of Graph Algorithms and Applications **7**(2), 131–140 (2003) http://www.cs.brown.edu/publications/jgaa/accepted/2003/Eppstein2003.7.2.pdf
11. Feder, T., Motwani, R.: Worst-case time bounds for coloring and satisfiability problems. J. Algorithms **45**(2), 192–201 (2002). https://doi.org/10.1016/S0196-6774(02)00224-9
12. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. Algorithmica **54**(2), 181–207 (2009)
13. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM **56**(5), 1–32 (2009). https://doi.org/10.1145/1552285.1552286
14. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Inf. Process. Lett. **97**(5), 191–196 (2006)
15. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer-Verlag, Berlin, Heidelberg (2010)
16. Fürer, M., Kasiviswanathan, S.P.: Algorithms for counting 2-Sat solutions and colorings with applications. In: proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM 2007), Lecture Notes in Computer Science, vol. 4508, pp. 47–57. Springer (2007) https://doi.org/10.1007/978-3-540-72870-2_5
17. Gaspers, S.: Exponential Time Algorithms. VDM Verlag, Saarbrucken, Germany (2010)
18. Gaspers, S., Lee, E.J.: Faster graph coloring in polynomial space. In: Computing and Combinatorics - 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings, pp. 371–383 (2017). https://doi.org/10.1007/978-3-319-62389-4_31
19. Gaspers, S., Sorkin, G.B.: A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between. J. Comput. Syst. Sci. **78**(1), 305–335 (2012). https://doi.org/10.1016/j.jcss.2011.05.010
20. Gaspers, S., Sorkin, G.B.: Separate, measure and conquer: Faster polynomial-space algorithms for max 2-csp and counting dominating sets. ACM Trans. Algorithms **13**(4), 44:1-44:36 (2017). https://doi.org/10.1145/3111499
21. Greenhill, C.: The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. Comput. Complex. **9**(1), 52–72 (2000)
22. Iwata, Y.: A faster algorithm for dominating set analyzed by the potential method. In: Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC 2011), Lecture Notes in Computer Science, vol. 7112, pp. 41–54. Springer (2011) https://doi.org/10.1007/978-3-642-28050-4_4
23. Junosza-Szaniawski, K., Tuczynski, M.: Counting independent sets via divide measure and conquer method. Tech. Rep. ArXiv: 1503.08323, arXiv CoRR (2015)
24. Koivisto, M.: An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 583–590. IEEE Computer Society (2006). https://doi.org/10.1109/FOCS.2006.11
25. Lawler, E.L.: A note on the complexity of the chromatic number problem. Inf. Process. Lett. **5**(3), 66–67 (1976)
26. Roth, D.: On the hardness of approximate reasoning. Artif. Intell. **82**(1), 273–302 (1996)
27. Scott, A.D., Sorkin, G.B.: Polynomial constraint satisfaction problems, graph bisection, and the ising partition function. ACM Transactions on Algorithms (TALG) **5**(4), 45 (2009)
28. Wahlström, M.: Exact algorithms for finding minimum transversals in rank-3 hypergraphs. J. Algorithms **51**(2), 107–121 (2004). https://doi.org/10.1016/j.jalgor.2004.01.001

29. Wahlström, M.: A tighter bound for counting max-weight solutions to 2SAT instances. In: Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC 2008), Lecture Notes in Computer Science, vol. 5018, pp. 202–213. Springer (2008) https://doi.org/10.1007/978-3-540-79723-4_19
30. Woeginger, G.J.: Open problems around exact algorithms. Discret. Appl. Math. **156**(3), 397–405 (2008). https://doi.org/10.1016/j.dam.2007.03.023