Serge Gaspers and Gregory B. Sorkin

Separate, measure and conquer: faster polynomial-space algorithms for Max 2-CSP and counting dominating sets

Article (Accepted version) (Refereed)

Separate, Measure and Conquer: Faster Polynomial-Space Algorithms for Max 2-CSP and Counting Dominating Sets

SERGE GASPERS, UNSW Sydney and Data61, CSIRO
GREGORY B. SORKIN, London School of Economics

We show a method resulting in the improvement of several polynomial-space, exponential-time algorithms. The method capitalizes on the existence of small balanced separators for sparse graphs, which can be exploited for branching to disconnect an instance into independent components. For this algorithm design paradigm, the challenge to date has been to obtain improvements in worst-case analyses of algorithms, compared with algorithms that are analyzed with advanced methods, notably Measure and Conquer. Our contribution is the design of a general method to integrate the advantage from the separator-branching into Measure and Conquer, for a more precise and improved running time analysis.

We illustrate the method with improved algorithms for Max $(r, 2)$-CSP and #Dominating Set. An instance of the problem Max $(r, 2)$-CSP, or simply Max 2-CSP, is parameterized by the domain size r (often 2), the number of variables n (vertices in the constraint graph G), and the number of constraints m (edges in G). When G is cubic, and omitting sub-exponential terms here for clarity, we give an algorithm running in time $r^{(1/5)\,n} = r^{(2/15)\,m}$; the previous best was $r^{(1/4)\,n} = r^{(1/6)\,m}$. By known results, this improvement for the cubic case results in an algorithm running in time $r^{(9/50)\,m}$ for general instances; the previous best was $r^{(19/100)\,m}$. We show that the analysis of the earlier algorithm was tight: our improvement is in the algorithm, not just the analysis. The same running time improvements hold for Max Cut, an important special case of Max 2-CSP, and for Polynomial and Ring CSP, generalizations encompassing graph bisection, the Ising model, and counting.

We also give faster algorithms for #Dominating Set, counting the dominating sets of every cardinality $0, \ldots, n$ for a graph G of order n. For cubic graphs, our algorithm runs in time $3^{(1/5)\,n}$; the previous best was $2^{(1/2)\,n}$. For general graphs, we give an unrelated algorithm running in time 1.5183^n; the previous best was 1.5673^n.

The previous best algorithms for these problems all used local transformations and were analyzed by the Measure and Conquer method. Our new algorithms capitalize on the existence of small balanced separators for cubic graphs — a non-local property — and the ability to tailor the local algorithms always to "pivot" on a vertex in the separator. The new algorithms perform much as the old ones until the separator is empty, at which point they gain because the remaining vertices are split into two independent problem instances that can be solved recursively. It is likely that such algorithms can be effective for other problems too, and we present their design and analysis in a general framework.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; **Parameterized complexity and exact algorithms**; *Backtracking*;

Additional Key Words and Phrases: Measure & Conquer, graph separators, Max 2-CSP, Counting dominating sets

Authors' addresses: S. Gaspers, School of Computer Science and Engineering, Building K17, UNSW Sydney, Sydney NSW 2052, Australia; G. B. Sorkin, Department of Mathematics, London School of Economics, Houghton Street, London WC2A 2AE, UK. .

1549-6325/2017/1-ART1 $15.00
https://doi.org/http://dx.doi.org/10.1145/3111499

ACM Reference format:
Serge Gaspers and Gregory B. Sorkin. 2017. Separate, Measure and Conquer: Faster Polynomial-Space Algorithms for Max 2-CSP and Counting Dominating Sets. *ACM Trans. Algor.* 1, 1, Article 1 (January 2017), 36 pages.
https://doi.org/http://dx.doi.org/10.1145/3111499

1 INTRODUCTION

1.1 Background and intuition of the method

Measure and Conquer [18] has become the prevalent method to analyze branching algorithms over the last few years. Several textbooks [13, 23] and PhD theses [4, 8, 9, 27, 36, 45, 59, 61, 66] give an extensive treatment of the method and examples of algorithms analyzed with Measure and Conquer. Many of these algorithms first branch on vertices of high degree before handling graphs with bounded maximum degree. Improvements in running times are often due to faster computations on small-degree graphs, and the Measure and Conquer method then allows to propagate this speed-up to higher degrees, for an overall improved running time.

Some general methods to obtain improvements in the running time lead to exponential-space algorithms. Among them are methods based on memoization (see, e.g., [37, 55]) and dynamic programming on tree decompositions or path decompositions (see, e.g., [17, 22]). In this paper, we aim to obtain comparable improvements as when performing dynamic programming on tree decompositions or path decompositions, but without using exponential space. Our method can be seen as exploiting the separation properties of path/tree decompositions in a branching process instead of dynamic programming. The design of fast exponential-time algorithms with restricted space usage has attracted attention for many problems, such as Steiner Tree [19, 51], Knapsack [48], Hamiltonian Path [39], and Coloring [5]. Further results and open problems surrounding the trade-off between time and space complexities are discussed in a survey by Woeginger [68].

The use of graph separators for divide-and-conquer algorithms dates back to the 1970s [47]. For classes of instances with sublinear separators, including those described by planar graphs, this often gives subexponential- or polynomial-time algorithms. It is natural to design a branching strategy that strives to disconnect an instance into components, even when no sublinear separators are known. While this has successfully been done experimentally (see, e.g., [3, 12, 25, 35, 44]), we are not aware of worst-case analyses of branching algorithms that are based on linear sized separators.[1]

Our algorithms exploit small separators, specifically, balanced separators of size about $n/6$ for cubic graphs of order n. The existence of such separators has been known since 2001 at least, they have been used in pathwidth-based dynamic programming algorithms using exponential time and exponential space, and it is natural to try to exploit them for algorithms running in exponential time but polynomial space.

Outline of the paper. (A more detailed outline follows as Section 1.2.) In the rest of the Introduction we introduce the separation results and discuss applying them to Max $(r, 2)$-CSP, or simply Max 2-CSP. In Max 2-CSP, the input is a graph, together with a score function for each vertex and each edge whose values depend on the color of the vertex or the endpoints of the edge, respectively. The task is to color the vertices with colors from $[r] = \{1, \ldots, r\}$ such that the sum of all scores is maximized. An important special case is Max Cut where $r = 2$ and the score functions always return 0, except when it is a score function of an edge and the two endpoints of the edge do not have the same color, in which case the score function returns 1. We will show how applying the

[1] Fomin et al. [20] exploit small balanced structures of a solution that is to be found in a graph, whereas our separators are separators for the input graph.

separation results naively, using the separator alone, gives an algorithm worse than existing ones for Max 2-CSP; perhaps this is why such algorithms have not already been developed. Then, we sketch how, under optimistic assumptions, using small separators *in conjunction with* existing Measure and Conquer analyses gives an improvement. In the body of the paper we turn this into a rigorous analysis of a faster algorithm for Max 2-CSP. We then describe a general framework, "Separate, Measure and Conquer", to design and analyze polynomial-space, exponential-time algorithms based on small balanced separators. We use it to obtain faster algorithms for #Dominating Set, calling upon the method in significantly greater generality than was needed for Max 2-CSP. We conclude with some thoughts on the method, including the likely limitation that the (polynomial-space) algorithms based on it will be slower than exponential-space dynamic programming algorithms based on path decompositions.

We now introduce the main separation results we will use. The *bisection width* of a graph is the minimum number of edges between vertices belonging to different parts, over all partitions of the vertex sets into two parts whose size differs by at most one. Monien and Preis [49, 50] proved the following upper bound on the bisection width of cubic graphs.

Theorem 1.1 ([50]). *For any $\epsilon > 0$ there is a value n_ϵ such that the bisection width of any cubic graph $G = (V, E)$ with $|V| > n_\epsilon$ is at most $(\frac{1}{6} + \epsilon)|V|$.*

As noted in [22], the bound also holds for graphs of maximum degree at most 3 and a corresponding bisection can be computed in polynomial time. Kostochka and Melnikov [42] showed that almost all cubic graphs on n vertices have bisection width at least $n/9.9$. To the best of our knowledge, this is the best known lower bound on the bisection width of cubic graphs.

Let (L, S, R) be a partition of the vertex set of a graph G such that there is no edge in G with one endpoint in L and the other endpoint in R. We say that (L, S, R) is a *separation* of G, and that S is a *separator* of G, *separating* L and R (thought of as Left and Right). The previous theorem immediately gives a small *balanced* separator, one partitioning the remaining vertices into equal-sized parts. It is obtained by choosing a vertex cover of the edges in the bisection.

Lemma 1.2. *For any $\epsilon > 0$ there is a value n_ϵ such that every graph $G = (V, E)$ of maximum degree at most 3, $|V| \geq n_\epsilon$, has a separation (L, S, R) with $|S| \leq (\frac{1}{6} + \epsilon)|V|$ and $|L|, |R| \leq \left\lceil \frac{|V|-|S|}{2} \right\rceil$. Moreover, such a separation can be computed in polynomial time.*

A direct application of branching on the vertices in such a separator yields algorithms inferior to existing Measure and Conquer ones. We illustrate with solving a cubic Max 2-CSP instance with n vertices. We separate the vertices as (L, S, R) per Lemma 1.2. Sequentially, we generate $r^{|S|}$ smaller instances by taking each possible assignment to the vertices of S. For Max 2-CSP, as for many problems, assigning a value to a vertex $v \in V$ yields an instance on vertices $V \setminus \{v\}$, so the procedure above produces instances on the vertex set $L \cup R$. The restriction of G to those vertices, $G|_{L \cup R}$, is by construction a graph with no edge between L and R, and the solution to an instance on constraint graph $G|_{L \cup R}$ is simply the direct combination of the solutions to the corresponding instances given by $G|_L$ and $G|_R$. The advantage given by reducing on vertices in the *separator* is that (for each branch) we must solve two small instances rather than one large one. Roughly speaking, in the worst case where $|S| = \frac{1}{6}n$ and $|L| = |R| = \frac{5}{12}n$, the running time $t(n)$ satisfies the recurrence

$$t(n) = r^{n/6}(2 \cdot t(\tfrac{5}{12}n)),$$

whose solution satisfies $t(n) = O^\star(r^{2n/7})$.[2] This is inferior to the $O^\star(r^{n/4})$ upper bound from [56, 57] for a simple polynomial-space algorithm based on local simplification and branching rules.

The improvements in this paper have their origin in a simple observation: if an algorithm can always branch on vertices in the separator, then the usual measure of improvement is achieved at each step (typically computed by the Measure and Conquer method described in Subsection 2.3), and the splitting of the graph into two parts when the separator is emptied is a bonus. We get the best of both. The technical challenges are to accurately amortize this bonus over the previous branches to prove a better running time, and to control the balance of the separation as the algorithm proceeds so that the bonus is significant.

Again, we illustrate our approach for cubic Max 2-CSP. As remarked earlier, we will be optimistic in this sketch, doing the analysis rigorously in Section 3. The problem class will also be defined there, but for now one may think of Max Cut, with domain size $r = 2$. Let us "pivot" on a vertex $v \in S$, i.e., sequentially assign it each possible value, eliminate it and its incident edges (see rule Reduction 3 below), and solve each case recursively. It is possible that v has one or more neighbors within S, but this is a favorable case, reducing the number of subsequent branches needed. So, suppose that v has neighbors only in L and R. If all its neighbors were in one part, the separator could be made smaller, so let us skip over this case as well. The cases of interest, then, are when v has two neighbors in L and one in R, or vice-versa. Suppose that these cases occur equally often; this is the bit of optimism that will require more care to get right. In that case, after all $|S|$ branchings, the sizes of L and R are each reduced by $\frac{3}{2}|S|$, since degree-2 vertices get contracted away. This would lead to a running time bound $t(n)$ satisfying the recurrence

$$t(n) = r^{n/6} \cdot 2t(\tfrac{5}{12}n - \tfrac{3}{2} \cdot \tfrac{1}{6}n),$$

leading to a solution with $t(n) = O^\star(r^{n/5})$. This conjectured bound would improve on the best previous algorithm's time bound of $O^\star(r^{n/4})$, and Section 3 establishes that the bound is true, modulo a subexponential factor in the running time due to Lemma 1.2 guaranteeing only $|S| \le (\frac{1}{6} + \epsilon)n$ instead of $|S| \le \frac{1}{6}n$.

Our algorithms exploit a *global* graph structure, the separator, while executing an algorithm based on *local* simplification and branching rules. The use of global structure may also make it possible to circumvent lower bounds for classes of branching algorithms that only consider local information when deciding whether to branch on a variable [1, 2].

1.2 Results and organization of the paper

Section 2 defines notation and gives the necessary background on separators and the Measure and Conquer method that are necessary for Section 3. Section 3 gives a first analysis of a separator-based algorithm. It solves cubic instances of Max 2-CSP in time $r^{(1/5+o(1))\,n}$, where n is the number of vertices. This improves on the previously fastest $O^\star\left(r^{n/4}\right)$ time polynomial-space algorithm [57]. The fastest known exponential-space algorithm for cubic instances uses the pathwidth approach of Fomin and Høie [22] to solve Max 2-CSP in time $r^{(1/6+o(1))\,n}$ [57].

Our algorithm also improves the fastest known running time for polynomial-space algorithms solving general instances of Max 2-CSP from $O^\star\left(r^{(19/100)\,m}\right)$ to $r^{(9/50+o(1))\,m}$, where m is the number of edges of the constraint graph. The same running time improvement holds for Max Cut, an important special case of Max 2-CSP, and for Polynomial and Ring CSP, generalizations encompassing graph bisection, the Ising model, and counting problems. For comparison, the fastest known exponential-space algorithm has worst-case running time $r^{(13/75+o(1))\,m}$ [57]. For vertex-parameterized running times, Williams' [67] exponential-space algorithm runs in time

[2] The notation $f(n) = O^\star(g(n))$ indicates that for some polynomial $p(n)$, for all sufficiently large n, $f(n) \le p(n)g(n)$.

$O^\star(()\,2^{(\omega/3)\,n})$, which is $O(1.7303^n)$ using the current best upper bound on the matrix-multiplication exponent ω [26].

Subsection 3.4 gives tight lower bounds for the analysis of the previously fastest polynomial-space Max 2-CSP algorithm. These lower bounds highlight that our new algorithm is strictly faster. It should be noted that tight analyses are rare for competitive branching algorithms. An important feature of these lower bounds is that they make the algorithm use several branching rules whose Measure and Conquer analyses are most constraining.

In independent work, Edwards [15] matches our running times for Max 2-CSP on general and cubic graphs. His work is also based on separators and is similar to ours from an algorithmic point of view. The running time analysis differs from ours in that it is targeted solely at Max 2-CSP, while we provide a general method, as we discuss next.

While Max 2-CSP is a central problem in exponential-time algorithms, the analysis of its branching algorithms is typically easier than for other problems, largely because the branching creates isomorphic subinstances. In Section 4, we develop the Separate, Measure and Conquer method in full generality, and use this in Section 5 to design faster polynomial-space algorithms for counting dominating sets in graphs. For graphs with maximum degree 3, we obtain an algorithm with a time bound of $3^{(1/5+o(1))\,n} = O(1.2458^n)$, improving on the previous best polynomial-space running time of $O^\star(2^{(1/2)\,n}) = O(1.4143^n)$ [41]. For general graphs, we obtain a different algorithm, with time bound $O(1.5183^n)$, improving on the previous best polynomial-space running time of $O(1.5673^n)$ [60]. For comparison, the fastest known exponential-space algorithms run in time $3^{(1/6+o(1))\,n}$ for cubic graphs (by combining [63] and [22]) and in time $O(1.5002^n)$ for general graphs [53].

Limitations of the Separate, Measure and Conquer method are discussed in Section 6, and we provide an outlook in the conclusion section.

The hasty reader may choose to skip Subsections 3.4, 5.1, and/or 5.2, or Section 5 altogether. A preliminary version of this article appeared in the proceeding of ICALP 2015 [31].

2 PRELIMINARIES

2.1 Graphs

Let $G = (V, E)$ be a (simple, undirected) graph, $v \in V$ be a vertex and $S \subseteq V$ a vertex subset of G. We also refer to the vertex set and edge set of G by $V(G)$ and $E(G)$, respectively. We denote the *open* and *closed neighborhoods* of v by $N_G(v) = \{u \in V : uv \in E\}$ and $N_G[v] = N_G(v) \cup \{v\}$, respectively. The *closed neighborhood* of S is $N_G[S] = \bigcup_{v \in S} N_G[v]$ and its *open neighborhood* is $N_G(S) = N_G[S] \setminus S$. The set S and vertex v are said to *dominate* the vertices in $N_G[S]$ and $N_G[v]$, respectively. The set S is a *dominating set* of G if $N_G[S] = V$. The degree of v in G is $d_G(v) = |N_G(v)|$. We sometimes omit the subscripts when they are clear from the context. The *contraction* of an edge $uv \in E$ is an operation replacing u and v by one new vertex c_{uv} that is adjacent to $N_G(\{u, v\})$. The graph $G - S$ is obtained from G by removing the vertices in S and all incident edges. When $S = \{v\}$ we also write $G - v$ instead of $G - \{v\}$. The subgraph induced on S is $G[S] = G - (V \setminus S)$. The graph G is *cubic* or 3*-regular* if each vertex has degree 3 and *subcubic* if each vertex has degree at most 3.

A *tree decomposition* of G is a pair $(\{X_i : i \in I\}, T)$ where each so-called *bag* $X_i \subseteq V$, $i \in I$, and T is a tree with elements of I as nodes such that:

(1) for each edge $uv \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$, and
(2) for each vertex $v \in V$, $T[\{i \in I : v \in X_i\}]$ is a (connected) tree with at least one node.

The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The *treewidth* [54] of G is the minimum width taken over all tree decompositions of G. *Path decompositions* and *pathwidth* are defined similarly, except that T is restricted to be a path.

2.2 Separators

Our algorithms will typically pivot (or branch) on a vertex of the separator S, separating L and R, producing instances with smaller separators. As reductions might render L much smaller than R, we would sometimes like to rebalance L and R without increasing the size of the separator.

Lemma 2.1. *Let $G = (V, E)$ be a subcubic graph and let S be a separator of G separating L and R. If there is a vertex $s \in S$ with exactly one neighbor l in L, then there is a separator S' of G separating L' and R' such that $|S'| = |S|$ and $|L'| = |L| - 1$.*

Proof. We use the partition $(L', S', R') = (L \setminus \{l\}, (S \setminus \{s\}) \cup \{l\}, R \cup \{s\})$. □

We say in this case that we *drag s* into R. This transformation, though it does not change the "true" problem instance, does change its presentation and its measure; it is treated like any other transformation in the Measure and Conquer analysis, to which we turn next.

2.3 Measure and Conquer

In this subsection, we recall the basics of the Measure and Conquer method. The method was introduced with that name by [18] but closely parallels Eppstein's quasi-convex method [16] and Scott and Sorkin's linear-programming approach [57]. We use the method in a form developed in [30], yielding convex mathematical programs.

To track the progress a branching algorithm makes when solving an instance, a Measure and Conquer analysis assigns a potential function to instances, a so-called measure.

Definition 2.2. A *measure* μ for a problem P is a function from the set of all instances for P to the set of non negative reals.

A Measure and Conquer analysis upper bounds the running time of a branching algorithm by analyzing the size of its search trees, which model the recursive calls the algorithm makes during an execution. Two values are of interest, an upper bound on the depth of the search trees and an upper bound on their number of leaves. In the following lemma, the measure η is typically polynomially bounded and used to bound the depth, and μ is used to bound the number of leaves. It is proved by a simple induction.

Lemma 2.3 ([27, 30]). *Let A be an algorithm for a problem P, $c \geq 0$ and $r > 1$ be constants, and $\mu(\cdot), \eta(\cdot)$ be measures for the instances of P, such that for any input instance I, A reduces I to instances $I_1, \ldots, I_k$, with $k \geq 1$, 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), with*

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

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

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

A reduction rule that creates at most one subinstance is a *simplification* rule, other reduction rules are *branching* rules.

3 MAX 2-CSP

Using the notation from [57], an instance $(G, \mathcal{S})$ of Max 2-CSP (also called Max $(r, 2)$-CSP) is given by a *constraint graph* $G = (V, E)$ and a set $\mathcal{S}$ of *score functions*. Writing $[r] = \{1, ..., r\}$ for the set of available vertex colors, we have a *dyadic* score function $s_e : [r]^2 \to \mathbb{R}$ for each edge $e \in E$, a *monadic* score function $s_v : [r] \to \mathbb{R}$ for each vertex $v \in V$, and a single *niladic* score "function" $s_\emptyset : [r]^0 \to \mathbb{R}$ which takes no arguments and is just a constant convenient for bookkeeping. A *candidate solution* is a function $\phi : V \longrightarrow [r]$ assigning colors to the vertices (ϕ is an *assignment* or *coloring*), and its score is

$$s(\phi) := s_\emptyset + \sum_{v \in V} s_v(\phi(v)) + \sum_{uv \in E} s_{uv}(\phi(u), \phi(v)). \tag{3}$$

An *optimal solution* ϕ is one which maximizes $s(\phi)$.

Let us recall the reductions from [57]. Reduction 0, 1, and 2 are simplification rules (acting, respectively, on vertices of degree 0, 1, and 2), creating one subinstance, and Reduction 3 is a branching rule, creating r subinstances. An optimal solution for $(G, \mathcal{S})$ can be found in polynomial time from optimal solutions of the subinstances.

Reduction 0 If $d(y) = 0$, then set $s_\emptyset = s_\emptyset + \max_{C \in [r]} s_y(C)$ and delete y from G.

Reduction 1 If $N(y) = \{x\}$, then replace the instance with $(G', \mathcal{S}')$ where $G' = (V', E') = G - y$ and $\mathcal{S}'$ is the restriction of $\mathcal{S}$ to V' and E' except that for all colors $C \in [r]$ we set

$$s'_x(C) = s_x(C) + \max_{D \in [r]} \{s_{xy}(C, D) + s_y(D)\}.$$

Reduction 2 If $N(y) = \{x, z\}$, then replace the instance with $(G', \mathcal{S}')$ where $G' = (V', E') = (V - y, (E \setminus \{xy, yz\}) \cup \{xz\})$ and $\mathcal{S}'$ is the restriction of $\mathcal{S}$ to V' and E', except that for $C, D \in [r]$ we set

$$s'_{xz}(C, D) = s_{xz}(C, D) + \max_{F \in [r]} \{s_{xy}(C, F) + s_{yz}(F, D) + s_y(F)\}$$

if there was already an edge xz, discarding the first term $s_{xz}(C, D)$ if there was not.

Reduction 3 Let y be a vertex of degree at least 3. There is one subinstance (G', s^C) for each color $C \in [r]$, where $G' = (V', E') = G - y$ and s^C is the restriction of s to V' and E', except that we set

$$(s^C)_\emptyset = s_\emptyset + s_y(C),$$

and, for every neighbor x of y and every $D \in [r]$,

$$(s^C)_x(D) = s_x(D) + s_{xy}(D, C).$$

We will now describe a new separator-based algorithm for cubic Max 2-CSP, outperforming the algorithm given by Scott and Sorkin in [57]. Using it as a subroutine in the algorithm for general instances by Scott and Sorkin [57] also gives a faster running time for Max 2-CSP on arbitrary graphs.

3.1 Background

For a cubic instance of Max 2-CSP, an instance whose constraint graph G is 3-regular, the fastest known polynomial-space algorithm makes simple use of the reductions above. The algorithm branches on a vertex v of degree 3, giving r instances with a common constraint graph G', where v has been deleted from the original constraint graph. In G', the three G-neighbors of v each have degree 2. Simplification rules are applied to rid G' of degree-2 vertices, and further vertices of degree 0, 1, or 2 that may result, until the constraint graph becomes another cubic graph G''. This results in r instances with the common constraint graph G'', to which the same algorithm is applied recursively. The running time of the algorithm is exponential in the number of branchings, and since each branching destroys four degree-3 vertices (the pivot vertex v and its three neighbors), the running time is bounded by $O^\star\left(r^{n/4}\right)$; details may be found in [56].

For this algorithm, $r^{n/4}$ is also a lower bound, achieved for example by an instance whose constraint graph consists of disjoint copies of K_4, the complete graph on 4 vertices. In fact, instances with disjoint constraint graphs can be solved with far greater efficiency, since the components can be solved separately, a fact exploited by the fastest polynomial-space algorithms for Max 2-CSP [57]. However, Subsection 3.4 shows that $r^{n/4}$ is also a lower bound for algorithms exploiting connectedness, since a slightly unlucky choice of pivot vertices leaves a particular worst-case constraint graph connected. We conjecture that $r^{n/4}$ is a lower bound for any algorithm using these reductions and choosing its pivot "locally": that is, characterizing each vertex by the structure of the constraint graph (or even the instance) within a fixed-radius ball around it, and choosing a vertex with a best such character.

Here, we show how to break this $r^{n/4}$ barrier, by selecting pivot vertices using global properties of the constraint graph. In this section we describe an algorithm which pivots only on vertices in a separator of G. When the separator is exhausted, G has been split into two components L and R which can be solved independently, and are treated recursively. The efficiency gain of the algorithm comes from the component splitting: if the time to solve an instance with n vertices can be bounded by $O^\star(r^{cn})$, the time to solve an instance consisting of components L and R is $O^\star\left(r^{c|L|}\right) + O^\star\left(r^{c|R|}\right)$, which (for L and R of comparable sizes) is hugely less than the time bound $O^\star\left(r^{c(|L|+|R|)}\right)$ for a single component of the same total order. This efficiency gain comes at no cost: until the separator is exhausted, branching on vertices in the separator is just as efficient as branching on any other vertex.

3.2 Algorithm and Analysis

We interleave the algorithm's description with its running time analysis. To analyze the algorithm, we use the Measure and Conquer method. As in [30], we use penalty terms in the measure to treat tricky cases that otherwise require arguments outside the Measure and Conquer framework. (Those arguments are typically simple but mesh poorly with the Measure and Conquer framework, making correctness difficult to check. Our penalty approach is modeled on Wahlström's [65].) We also take from [57] and [30] the treatment of vertices of degrees 1 and 2 within the Measure and Conquer framework.Reductions on such vertices are *de facto* never an algorithm's critical cases, but can lead to a tangle of special cases unless treated uniformly.

Recall (see Lemma 2.3) that the Measure and Conquer analysis applies to an algorithm which transforms an instance I to one or more instances $I_1, \ldots, I_k$ in polynomial, solves those instances recursively, and obtains a solution to I in polynomial time from the solutions of $I_1, \ldots, I_k$. By definition, the measure $\mu(I)$ of an instance I should satisfy that for any instance,

$$\mu(I) \geq 0. \tag{4}$$

To satisfy constraint (1) of Lemma 2.3 we should have that for any transformation of I into $I_1, \ldots, I_k$,

$$r^{\mu(I_1)} + \cdots + r^{\mu(I_k)} \leq r^{\mu(I)}. \tag{5}$$

Given these hypotheses, the algorithm solves any instance I in time $O^\star\left(r^{\mu(I)}\right)$ if the number of recursive calls from the root to a leaf of the search tree is polynomially bounded.

Here, we present an instance of Max 2-CSP in terms of a separation (L, S, R) of its constraint graph $G = (V, E)$. We write L_3, S_3, and R_3 for the subsets of degree-3 vertices of L, S, and R, respectively, and we will always assume that $|L_3| \leq |R_3|$, if necessary swapping the roles of L and R to make it so. We write $|S_2|$ for the number of degree-2 vertices in S.

We define the measure of an instance as

$$\mu(L, S, R) = w_s|S_3| + w_{s,2}|S_2| + w_r|R_3| + w_b\mathbb{1}(|R_3| = |L_3|) + w_c\mathbb{1}(|R_3| = |L_3| + 1) + w_d \log_{3/2}(|R_3| + |S_3|), \quad (6)$$

where the values w_s, $w_{s,2}$, w_r, w_b, w_c, and w_d are constants to be determined and the indicator function $\mathbb{1}$(event) takes the value 1 if the event is true and 0 otherwise. In what follows, we describe the algorithm and derive which constraints inequality (5) induces. Notice that if we choose the values w_s, $w_{s,2}$, w_r, w_b, w_c, and w_d so that all the constraints are satisfied, then Lemma 2.3 implies an algorithm running in time $O^\star\left(r^{\mu(I)}\right)$. We optimize $\mu(I)$ under the derived constraints.ÂŤ

For the constraint (4) that $\mu \geq 0$, it suffices to constrain each of the constants to be nonnegative:

$$w_s, w_{s,2}, w_r, w_b, w_c, w_d \geq 0. \quad (7)$$

Intuitively, the terms w_b and w_c are the only representations of the size of L in μ, and account for the greater time needed when the left side is as large (or nearly as large) as the right. (We thus anticipate that optimally setting the weights gives $w_b \geq w_c$, as turns out to be true, due to constraint (8).) The logarithmic term offsets increases in penalty terms that may result when a new separator is computed, where the instance may go from imbalanced to balanced.

Concretely, from (5), each reduction imposes a constraint on the measure. Whenever there is a vertex of degree 0, 1, or 2 a corresponding reduction is applied, so it can be presumed that any other reduction is applied to a cubic graph. We treat the reductions in their order of priority: when presenting one reduction, we assume that no previousy presented reduction can be applied to the instance. Denote by μ the value of the measure before the reduction is applied and by μ' its value after the reduction.

Degree 0. If the instance contains a vertex v of degree 0, then perform Reduction 0 on v. Removing v from the instance has no effect on the measure and Condition (5) is satisfied.

Half-edge deletion. If there is a vertex of degree 1, apply Reduction 1 on it, and if there is a vertex in $L \cup R$ of degree 2, apply Reduction 2 on it. We may view the deletion of an edge uv as a deletion of two half-edges, one incident on u and the other incident on v. Deleting the half-edge incident on v also decreases the degree of v, but not the degree of u. Similarly, when a reduction creates an edge between two vertices that already had an edge, our analysis considers that a temporary parallel edge emerges, which then collapses into a single edge.

We will require that a half-edge deletion does not increase the measure. This will also ensure that the collapse of parallel edges, Reduction 1, and Reduction 2 for vertices in $L \cup R$ does not increase the measure, which also validates Condition (5) for Reduction 1, and Reduction 2 for vertices in $L \cup R$. The only way that a half-edge deletion affects the measure is that the degree of a vertex v decreases.

First, assume $d(v) = 1$. The degree of v is reduced to 0. Since neither degree-0 nor degree-1 vertices affect the measure, $\mu' - \mu = 0$, satisfying Condition (5).

Next, assume $d(v) = 2$. If $v \in L$, then $\mu' - \mu = 0$. If $v \in S$, then $\mu' - \mu \leq -w_{s,2}$, which satisfies Condition (5) since $w_{s,2} \geq 0$ by (7). If $v \in R$, then $\mu' - \mu = 0$, which also satisfies Condition (5).

Finally, assume $d(v) = 3$. We consider three cases.

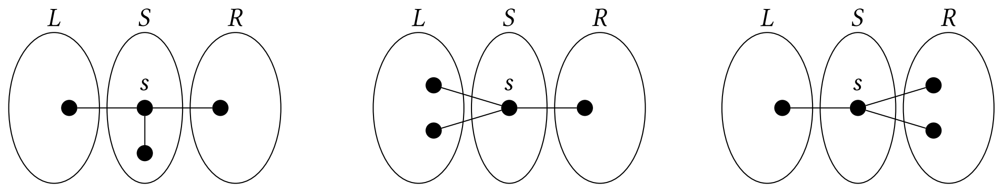
- $v \in L$. Since the imbalance increases by one vertex, $\mu' - \mu \leq \max(0, -w_c, -w_b + w_c)$. Since $w_c \geq 0$ by (7) it suffices to constrain

$$-w_b + w_c \leq 0. \quad (8)$$

- $v \in S$. For this case it is sufficient that
$$-w_s + w_{s,2} \le 0. \tag{9}$$
- $v \in R$. The case where $|R_3| = |L_3|$ is covered by (8) since we can swap L and R. Otherwise, $|R_3| \ge |L_3| + 1$, and then $\mu' - \mu \le -w_r + \max(0, w_c, w_b - w_c)$. Since $w_r \ge 0$ by (7), it suffices to constrain
$$-w_r + w_c \le 0 \text{ and} \tag{10}$$
$$-w_r + w_b - w_c \le 0. \tag{11}$$

Separation. This reduction is the only one special to separation, and its constraint looks quite different from those in previous works. The reduction applies when $S = \emptyset$, which arises in two cases. One is at the beginning of the algorithm, when the instance has not been separated, and may be represented by the trivial separation $(\emptyset, \emptyset, V)$. The second is when reductions on separated instances have exhausted the separator, so that S is empty but L and R are nonempty, and the instance is solved by solving the instances on L and R independently, via a new separation (L', S', R') for R and another such separation (L'', S'', R'') for L. The reduction is applied to a graph $G = (V, E)$ that is cubic and can be assumed to be of at least some constant order, $|V| \ge k$, since a smaller instance can be solved in constant time. By Lemma 1.2 we know that, for any constant $\epsilon > 0$, there is a size $k = k(\epsilon)$ such that any cubic graph G of order at least k has a separation (L, S, R) with $|S| \le (\frac{1}{6} + \epsilon)|V|$, $|L|, |R| \le \frac{5}{12}|V|$. To satisfy (5), we will make worst-case assumptions about balance, namely that the instance goes from being imbalanced ($|R_3| \ge |L_3| + 2$) to being balanced ($|R_3'| = |L_3'|$ and $|R_3''| = |L_3''|$). (Note that $w_b \ge w_c \ge 0$ by (7) and (8).) It thus suffices to constrain that
$$r^{w_s|S_3'| + w_r|R_3'| + w_b + w_d \log_{3/2}(|R_3'| + |S_3'|)} + r^{w_s|S_3''| + w_r|R_3''| + w_b + w_d \log_{3/2}(|R_3''| + |S_3''|)} \le r^{w_r|R_3| + w_d \log_{3/2}(|R_3|)}.$$
From the separator properties, this in turn is implied by
$$2 \cdot r^{w_s(1/6+\epsilon)|R_3| + w_r(5/12 \cdot |R_3|) + w_b + w_d \log_{3/2}(8/12 \cdot |R_3|)} \le r^{w_r|R_3| + w_d \log_{3/2}(|R_3|)},$$
where we have estimated $|L_3'|, |R_3'| \le \frac{5}{12}|R_3|$ and $|S_3'| \le (\frac{1}{6} + \epsilon)|R_3| \le \frac{3}{12}|R_3|$ in the log term on the left hand side. Since $r \ge 2$, it suffices to constrain that
$$1 + w_s(\tfrac{1}{6} + \epsilon)|R_3| + w_r(\tfrac{5}{12}|R_3|) + w_b + w_d \ \log_{3/2}(\tfrac{8}{12}|R_3|) \le w_r|R_3| + w_d \ \log_{3/2}(|R_3|).$$
Since we took $\frac{3}{2} = \frac{12}{8}$ to be the logarithm's base, we set $w_d = w_b + 1$ so that the term $w_d \log_{3/2}(\frac{8}{12}|R_3|)$ is equal to $-(w_b + 1) + (w_b + 1)\log_{3/2}(|R_3|)$, and it suffices to have
$$(\tfrac{1}{6} + \epsilon)w_s + \tfrac{5}{12}w_r \le w_r. \tag{12}$$

Degree 2 in** S**. If the instance has a vertex $s \in S$ of degree 2, then perform Reduction 2 on s. Let u_1, u_2 denote the neighbors of s. The vertex s is removed and the edge u_1u_2 is added if it was not present already. If L or R contain no neighbor of s, Condition (5) is implied by the constraints of the half-edge deletions. If $u_1 \in L$ and $u_2 \in R$ (or the symmetric case), then S is not a separator any more. The algorithm removes u_2 from R and adds it to S. If $d(u_2) = 2$, we have that $\mu' - \mu \le 0$. Otherwise, $d(u_2) = 3$. If initially we had $|L_3| = |R_3|$, then L and R will be swapped after the reduction, and we constrain that
$$-w_{s,2} + w_s - w_b + w_c \le 0. \tag{13}$$



(a) One neighbor in L, S, and R (b) Two neighbors in L (c) Two neighbors in R

Fig. 1. Configurations for branching on a separator vertex.

Otherwise, $\mu' - \mu \leq -w_{s,2} + w_s - w_r + \max(0, w_c, w_b - w_c)$. Since $w_c \geq 0$ by (7) it suffices to constrain

$$-w_{s,2} + w_s - w_r + w_c \leq 0, \text{ and} \tag{14}$$

$$-w_{s,2} + w_s - w_r + w_b - w_c \leq 0. \tag{15}$$

In the remaining cases, every vertex has degree 3.

No neighbor in L. If the separation (L, S, R) has a vertex $v \in S$ with no neighbor in L, "drag" v into R, i.e., transform the instance by changing the separation to $(L', S', R') := (L, S \setminus \{v\}, R \cup \{v\})$. It is easily checked that this is a valid separation, with no edge incident on both L and R, and with $|L_3'| \leq |R_3'|$ implied by $|L_3| \leq |R_3|$. Indeed the new instance is no more balanced than the old, so that the difference between the new and old measures is $\mu' - \mu \leq -w_s + w_r$, and to satisfy condition (5) it suffices that

$$-w_s + w_r \leq 0, \tag{16}$$

since by (7) and (8) an increase in imbalance does not increase the measure.

No neighbor in R. This case is similar to the previous case, but a vertex $v \in S$ with no neighbor in R is dragged into L.

- If initially we had $|R_3| = |L_3|$, we reverse the roles of L and R and revert to the previous case.
- If $|R_3| \geq |L_3| + 1$ then the transformation increases $|L_3|$ by 1, decreasing the imbalance by one vertex. Therefore, $\mu' - \mu \leq -w_s + \max(0, w_c, w_b - w_c)$ and we constrain that

$$-w_s + w_c \leq 0, \text{ and} \tag{17}$$

$$-w_s + w_b - w_c \leq 0. \tag{18}$$

With the above cases covered, we may assume that the pivot vertex $s \in S$ has degree 3 and at least one neighbor in each of L and R.

One neighbor in each of L***,*** S***, and*** R. If there is a vertex $s \in S$ with one neighbor in each of L, S, and R (Figure 1a), perform Reduction 3 on s, deleting it from the constraint graph and thereby reducing the degree of each neighbor to 2. Since both L and R lose a degree-3 vertex, there is no change in balance and the constraint is

$$1 - 2w_s + w_{s,2} - w_r \leq 0. \tag{19}$$

The form and the initial 1 come from the reduction's generating r instances with common measure μ', so the constraint is $r \cdot r^{\mu'} \leq r^{\mu}$, or equivalently $1 + \mu' - \mu \leq 0$. The value of $\mu' - \mu$ comes from S losing two degree-3 vertices but gaining a degree-2 vertex, and R losing a degree-3 vertex.

Two neighbors in L. If $s \in S$ has two neighbors in L and one neighbor in R (Figure 1b), applying Reduction 3 removes s, reduces the degree of a degree-3 vertex in R, and increases the imbalance by one vertex. The algorithm performs Reduction 3 if $|R_3| \leq |L_3| + 1$, where $\mu' - \mu \leq -w_s - w_r + \max(-w_b + w_c, -w_c)$. Thus, we constrain

$$1 - w_s - w_r - w_b + w_c \leq 0 \text{ and} \tag{20}$$
$$1 - w_s - w_r - w_c \leq 0. \tag{21}$$

If, instead, $|R_3| \geq |L_3|+2$, then the algorithm drags s into L and its neighbor $r \in R$ into S, replacing (L, S, R) by $(L \cup \{s\}, (S \setminus \{s\}) \cup \{r\}, R \setminus \{r\})$. We need to ensure that $-w_r + \max(w_b, w_c) \leq 0$, which, since $w_c \leq w_b$ by (8), is satisfied if we constrain that

$$-w_r + w_b \leq 0. \tag{22}$$

Two neighbors in R. If $s \in S$ has two neighbors in R and one neighbor in L (Figure 1c), the algorithm performs Reduction 3, which removes s, reduces the degree of two degree-3 vertices in R, and decreases the imbalance by one. For the analysis of the case where $|R_3| = |L_3|$, we refer to (20) since L and R are swapped after the reduction. For the other cases, we constrain

$$1 - w_s - 2w_r - w_c + w_b \leq 0 \text{ and} \tag{23}$$
$$1 - w_s - 2w_r + w_c \leq 0. \tag{24}$$

This concludes the description of the algorithm and describes all the constraints on the measure. To minimize the running time proven by the analysis (see after (4), (5)), we minimize the initial measure, which is $\mu(\emptyset, \emptyset, V) = w_r|V| + w_d \log_{3/2}(|V|)$. Since the logarithmic term affects the running time only by a polynomial, we therefore minimize w_r, subject to our constraints (7)–(24), which are all linear. We obtain the following optimal, feasible weights using linear programming:

$$w_r = 0.2 + o(1) \qquad w_s = 0.7 \qquad w_b = 0.2$$
$$w_{s,2} = 0.6 \qquad w_c = 0.1$$

All constraints are satisfied and $\mu = (1/5 + o(1))n$. The tight constraints in our analysis are (12)–(15) and (19)–(24).

It only remains to verify that the depth of the search trees of the algorithm is upper bounded by a polynomial. Since not every reduction removes a vertex (some only modify the separation (L, S, R)), it is crucial to guarantee some kind of progress for each reduction. We will argue that each reduction decreases another polynomially-bounded measure $\eta(L, S, R, E) := 3|S_3| + 2|R_3| + |L_3| + 2|E|$, by at least one, and the depth of the search trees is therefore polynomial. For those reductions that remove one or more vertices, it is easily seen that $\eta(L, S, R, E)$ decreases by at least one. In the Separation case, the instance $(L, \emptyset, R)$ leads to two instances $(\emptyset, \emptyset, L)$ and $(\emptyset, \emptyset, R)$ in a first step, and for each of these two instances, the measure decreases by at least one, since $|R_3| \geq |L_3|$. In a second step, an instance $(\emptyset, \emptyset, R)$ is replaced by (L', S', R') where the measure decreases by at least one since $|L'| > |S'|$. For those reductions where only the partition (L, S, R) is modified, it suffices to note that either a vertex is moved from S to $L \cup R$ (in the cases No neighbor in L and No neighbor in R) or that a degree-3 vertex is moved from R to S and a degree-3 vertex is moved from S to L (Two neighbors in L with $|R_3| \geq |L_3| + 2$).

3.3 MAX 2-CSP result, consequences, and extensions

The previous section established that our algorithm solves cubic instances of Max 2-CSP in time $r^{n/5+o(n)}$. The algorithm uses only polynomial space, since it is recursive with polynomial recursion depth and uses polynomial space in each recursive call.

The polynomial-space Max 2-CSP algorithm described in [57], for solving an arbitrary instance with n vertices and m edges in time $O^\star(r^{19m/100})$, relied upon the ability to solve cubic cases in time $O^\star(r^{m/6})$. A theorem in the same work, [57, Theorem 22], quantifies how speedups in solving the cubic case translate into speedups for general instances and for instances with constraint graphs of maximum degree 4, and shows that if the cubic solver is polynomial-space then so is the general algorithm. For instance, for maximum degree 4 graphs, Theorem 22 of [57] states that an algorithm for cubic graphs with running time $O^*(r^{\alpha m})$ leads to an algorithm for maximum degree 4 graphs with running time $O^*(r^{\beta_4(\alpha)m})$, where

$$\beta_4(\alpha) = \begin{cases} 1/8 + (3/8)\alpha & 1/9 \le \alpha \le 1/5 \\ 1/6 & 0 \le \alpha \le 1/9. \end{cases}$$

Thus, our cubic algorithm's running time of $O^*(r^{(2/15+\epsilon)m})$, for any $\epsilon > 0$, leads to an algorithm with running time $O^*(r^{(7/40+\epsilon')m})$ for arbitrarily small $\epsilon' > 0$ for maximum degree 4 graphs.

Theorem 3.1. *On input of a Max 2-CSP instance on a constraint graph G with n vertices and m edges, the described algorithm solves G in time $r^{n/5+o(n)} = r^{2m/15+o(m)}$ if G is cubic, time $r^{7m/40+o(m)}$ if G has maximum degree 4, and time $r^{9m/50+o(m)}$ in general, in all cases using only polynomial space.*

This improves respectively on the previous best polynomial-space running times of $O^\star(r^{m/6})$, $O^\star(r^{3m/16})$, and $O^\star(r^{19m/100})$, all from [57]. It also improves on the fastest known polynomial-space running times for Max Cut on cubic, maximum degree 4, and general graphs. The Max Cut problem, a special case of Max $(2,2)$-CSP, is to find a bipartition of the vertices of a given graph maximizing the number of edges crossing the bipartition. Its previously fastest polynomial-space algorithms were the one by Scott and Sorkin [57] for cubic, maximum degree 4, and general graphs.

Theorem 3.1 extends instantly to generalizations of Max 2-CSP introduced by Scott and Sorkin in [58]: Ring CSP (RCSP), where the scores take values in an arbitrary ring, and Polynomial CSP (PCSP), where the scores are multivariate formal polynomials. In analogy with the definition of a Max 2-CSP instance in and around (3), an instance $I = (G, \mathcal{S})$ of RCSP over a ring R and domain $[r]$ is composed of a graph $G = (V, E)$ and a set $\mathcal{S}$ of score functions: a *dyadic* score function $s_e : [r]^2 \to R$ for each edge $e \in E$, a *monadic* score function $s_v : [r] \to R$ for each vertex $v \in V$, and a *niladic* score function $s_\emptyset : [r]^0 \to R$. The score of an assignment $\phi : V \to [r]$ is the ring element

$$s(\phi) := s_\emptyset \cdot \prod_{v \in V} s_v(\phi(v)) \cdot \prod_{uv \in E} s_{uv}(\phi(u), \phi(v)),$$

and the RCSP problem is to compute the *partition function*

$$Z_I = \sum_{\phi: V \to [r]} s(\phi).$$

Comparing with Max 2-CSP, in RCSP the scores are ring-valued (rather than real-valued), the scores are multiplied (rather than added), and the solution is the sum of all assignment scores (rather than the maximum). (In fact, for our purposes the "ring" in RCSP can be relaxed to a semiring, like a ring but lacking negation, and Max 2-CSP may be viewed as the case of the semi-ring R over the reals, where the product operation for R is the real sum, the addition operation for R is the maximum of the two reals, and the zero for R is $-\infty$.)

Scott and Sorkin [58] showed that any branching algorithm for Max 2-CSP based solely on Reductions 0, 1, 2, and 3 can be extended to RCSP. They replace Reductions 0, 1, 2, and 3 by reductions appropriate for the more general setting and prove that each one can be executed using $O(r^3)$ ring operations, while changing the graph in the same way as the original reductions. The number of ring operations is thus of the same order as the number of instances in the Max 2-CSP branching algorithm. Where Theorem 4 of [58] applied this observation to "Algorithm B" of [57], applying the observation to our algorithm yields the following theorem.

Theorem 3.2. *Let R be a ring. On input of an RCSP instance I over R with domain $[r]$, n variables, m constraints, and constraint graph G, the ring extension of the algorithm described here calculates the partition function Z_I of I in polynomial space and with $r^{n/5+o(n)}$ ring operations if G is cubic, $r^{7m/40+o(m)}$ ring operations if G has maximum degree 4, and $r^{9m/50+o(m)}$ ring operations in general.*

[58] defines PCSP as the special case of RCSP where the ring R is a polynomial ring over the reals. The polynomials may be multivariate, and indeed may have negative and fractional powers. The obvious use of this is for generating functions. For example, to count the number of cuts a_{ij} of a graph with i vertices in side "0" and j cut edges, take the score of each vertex v to be x^0 if $\phi(v) = 0$ and x^1 if $\phi(v) = 1$, and the score of each edge (u, v) to be y^0 if $\phi(u) = \phi(v)$ and y^1 if $\phi(u) \neq \phi(v)$; then the partition function $Z = \sum a_{ij} x^i y^j$ gives all the desired counts a_{ij}. For further details and other applications see [58]. As a special case of RCSP, PCSP can of course be solved by the same algorithm in the same number of ring operations, but here our interest is in the total running time, so the times to multiply and add the polynomials must be taken into account. The details of this are beyond the scope of this article, but [58] defines "polynomially bounded" PCSP instances, where the ring operations can be performed efficiently, and a more general class of "prunable" PCSP instances, where the leading term of the partition function can be computed efficiently (even though the full polynomial may not be efficiently computable, and may for example be of size exponential in the input size). In analogy with [58, Theorem 12], this yields the following theorem.

Theorem 3.3. *The PCSP extension of our algorithm solves any polynomially bounded PCSP instance (or finds the pruned partition function of any prunable PCSP instance) with domain $[r]$, n variables, m constraints, and constraint graph G, in polynomial space and in time $r^{n/5+o(n)}$ if G is cubic, time $r^{7m/40+o(m)}$ if G has maximum degree 4, and time $r^{9m/50+o(m)}$ in general.*

Consequences include the ability to efficiently solve graph bisection, count Max r-SAT solutions, find judicious partitions, and compute the Ising partition function; the formulation of these and other problems as PCSPs is given in [58].

Edwards [15] derived, using a result from [14], a general way to design fast algorithms in terms of average degree based on algorithms whose running times depend on the number of edges of the graph. In particular, the running time for general graphs from Theorem 3.1 (which is matched by Edwards [15]) leads to a polynomial-space algorithm solving Max 2-CSP instances with average degree $d \geq 2$ in time $O^\star\left(r^{n \cdot \left(1 - \frac{3.4}{d+1} + O(1/d^3)\right)}\right)$ and polynomial space.

Theorem 3.4 ([15]). *Max 2-CSP instances with average degree $d \geq 2$ can be solved in time $O^\star\left(r^{n \cdot \left(1 - \frac{3.4}{d+1} + O(1/d^3)\right)}\right)$ and polynomial space.*

Finally, we would like to highlight that any improvement on Lemma 1.2 will automatically improve our running times.

3.4 Lower Bounds for the Scott-Sorkin Algorithm

Scott and Sorkin [57] analyzed the running time of their Max 2-CSP algorithm with respect to m, the number of edges of the input instance, and showed the following upper bounds:

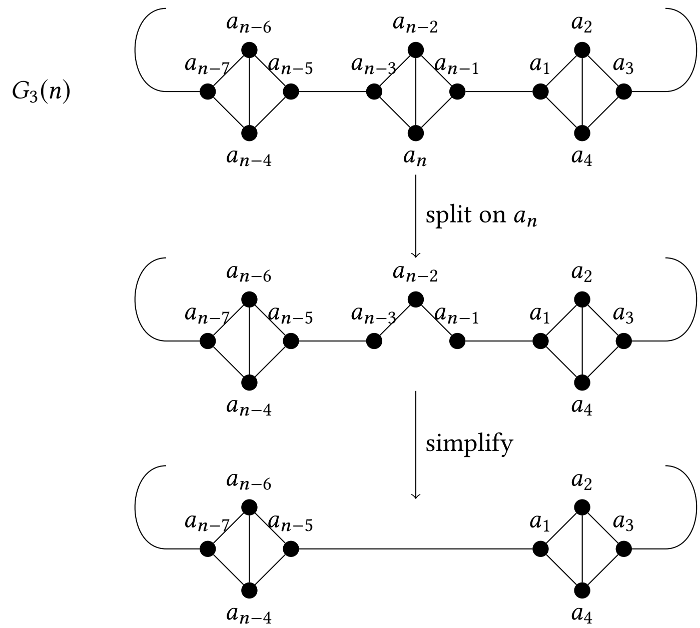


Fig. 2. The constraint graph $G_3(n)$ and how it evolves when branching on vertex a_n.

- $O^\star\left(r^{m/6}\right)$ for subcubic instance,
- $O^\star\left(r^{3m/16}\right)$ for instances with maximum degree 4, and
- $O^\star\left(r^{19m/100}\right)$ for instances with no degree restrictions.

In this subsection we prove matching lower bounds. Define $f(n) = \Theta^\star(g(n))$ if $f(n) = O^\star(g(n))$ and $f(n) = \Omega(g(n))$. By Lemmas 3.6–3.8, proved hereafter, and the analysis from [57], we obtain the following theorem.

Theorem 3.5. *The worst-case running time of the Scott-Sorkin algorithm for Max 2-CSP is*

- $\Theta^\star\left(r^{m/6}\right)$ *for subcubic instances,*
- $\Theta^\star\left(r^{3m/16}\right)$ *for instances with maximum degree 4, and*
- $\Theta^\star\left(r^{19m/100}\right)$ *for instances with no degree restrictions,*

where m is the number of edges in the input instance, even when the input instance is connected.

The lower bounds are established by Lemmas 3.6–3.8, starting with instances with maximum degree 3. These lemmas define infinite families of instances with maximum degrees 3, 4, and 5, and prove that the algorithm may perform $\Omega(r^{m/6})$, $\Omega(r^{3m/16})$, and $\Omega(r^{19m/100})$ steps, respectively. From the dual solution in the LP analysis in [57], it follows that the $r^{19m/100}$ time bound for general instances rests on the algorithm branching on vertices of degrees 5, 4, and 3 in the proportion 8 to 6 to 5, and that no additional Reduction 0, 1, and 2 occur beyond the ones directly needed by Reduction 3 (an application of Reduction 3 on a vertex v necessarily leads to an application of Reduction 0, 1, or 2 on each neighbor of v). The LP analysis does not indicate how to go about constructing instances for which this occurs, nor prove their existence. The running time lower bound of Lemma 3.8 is proved by explicitly constructing such instances.

Lemma 3.6. *On connected subcubic instances, the Scott-Sorkin algorithm has worst-case running time $\Omega(r^{m/6})$.*

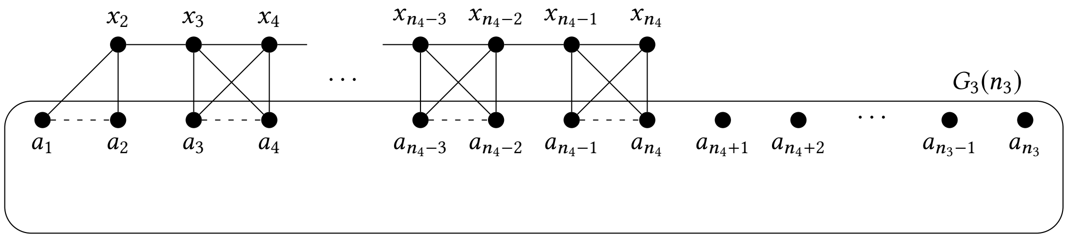


Fig. 3. The graph $G_4(n_3, n_4)$, with n_3 divisible by 4 and $n_4 \leq n_3$ divisible by 2, is obtained from $G_3(n_3)$ by removing the matching $\{a_1a_2, \ldots, a_{n_4-1}a_{n_4}\}$ (dashed edges), and adding $n_4 - 1$ new vertices $x_2, x_3, \ldots, x_{n_4}$ and the depicted edges.

Proof. The lemma is proven by exhibiting an infinite family of connected 3-regular instances such that the algorithm may perform $r^{n/4}$ steps when given an instance on n vertices from this family.

Let n be an integer that is divisible by 4. Consider the execution of the Scott-Sorkin algorithm on a Max 2-CSP instance whose constraint graph is the graph $G_3(n)$ from Figure 2. It is obtained from a cycle $(a_1, a_2, a_3, a_5, a_6, a_7, \ldots, a_{n-3}, a_{n-2}, a_{n-1})$ on $3n/4$ vertices by adding $n/4$ new vertices $a_4, a_8, \ldots, a_n$ and the edges $a_{4i}a_{4i-3}, a_{4i}a_{4i-2}, a_{4i}a_{4i-1}, 1 \leq i \leq n/4$.

The algorithm selects an arbitrary vertex of degree 3 and splits on it (i.e., it performs Reduction 3 on it). Suppose the algorithm selects a_n.

Let us first show that the constraint graph that is obtained by performing Reduction 3 on a_n and simplifying the instance is $G_3(n-4)$. If $n = 4$, observe that $G_3(4)$ is a complete graph on 4 vertices. Branching on one vertex leaves a complete graph on 3 vertices which vanishes by applications of Reduction 2, Reduction 1 and Reduction 0. If $n > 4$, the algorithm splits on variable a_n, which creates r instances where a_n is removed. Afterwards, Reduction 2 is performed on the variables a_{n-3}, a_{n-2}, and a_{n-1}, and the constraint graph of the resulting instances is $G_3(n-4)$.

Thus, Reduction 3 is executed $n/4 = m/6$ times recursively by the algorithm, resulting in a running time of $\Omega(r^{m/6})$. □

Lemma 3.7. *On connected instances with maximum degree 4, the Scott-Sorkin algorithm has worst-case running time* $\Omega(r^{3m/16})$.

Proof. The lemma is proven by exhibiting an infinite family of connected instances with maximum degree 4, where only a constant number of vertices have degree less than 4, such that the algorithm may perform $\Omega(r^{3n/8})$ steps when given an instance on n vertices from this family.

The graph family will use the following construction. The graph $G_4(n_3, n_4)$, with n_3 divisible by 4 and $n_4 \leq n_3$ divisible by 2, is obtained from $G_3(n_3)$ by the following modifications. Let $M = \{a_1a_2, a_3a_4, \cdots, a_{n_4-1}a_{n_4}\}$ and observe that M is a matching in $G_3(n_3)$. Remove the edges in M from the graph. Add a path $(x_2, x_3, \ldots, x_{n_4})$ on $n_4 - 1$ new vertices to the graph, and add the edges $x_ia_i, 2 \leq i \leq n_4$, the edge x_2a_1, and the edges $x_ia_{i-1}, x_{i-1}a_i$, for all even $i = 4, 6, \ldots, n_4$. See Figure 3.

On graphs with maximum degree 4, the Scott-Sorkin algorithm performs Reduction 3 on vertices of degree 4, with a preference for those vertices of degree 4 that have neighbors of degree 3. Assume that $n_4 \geq 4$ and that the algorithm splits on x_{n_4-1}. We claim that this creates two instances, both with constraint graph $G_4(n_3, n_4 - 2)$. Indeed, branching on x_{n_4-1} creates a constraint graph where x_{n_4-1} is removed, which triggers Reduction 2 on x_{n_4}, resulting in $G_4(n_3, n_4 - 2)$. If $n_4 = 2$, then Reduction 2 applies to x_2, creating $G_4(n_3, 0) = G_3(n_3)$.

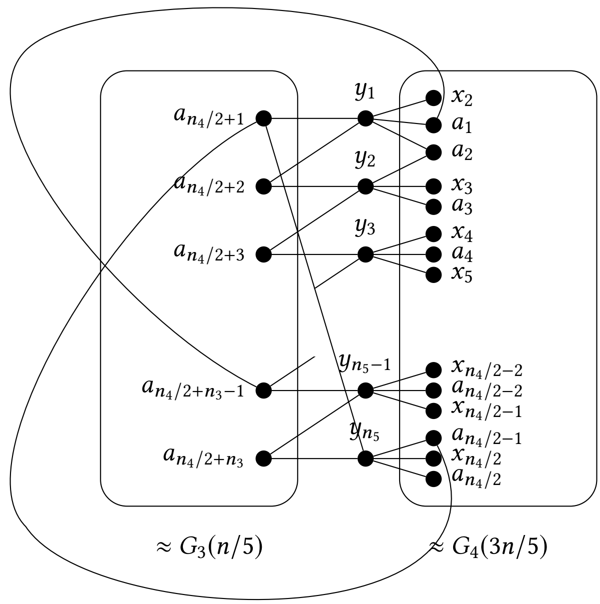


Fig. 4. The graph $G_5(n)$ is obtained from $G_4(n/2, 3n/10)$ by adding an independent set $\{y_1, \dots, y_{n/5}\}$ and the depicted edges.

We conclude that Reduction 3 is executed $n_4/2 - 2$ times recursively by the algorithm before reaching instances with constraint graph $G_3(n_3)$, for which the running time is characterized by Lemma 3.6. The algorithm may therefore execute

$$r^{n_4/2-2} \cdot r^{n_3/4} \tag{25}$$

steps. Setting $n_3 = n_4$, we obtain a running time of $\Omega(r^{(n/2-1)/2-2} \cdot r^{n/8}) = \Omega(r^{3n/8}) = \Omega(r^{3m/16})$. □

Lemma 3.8. *On connected instances with maximum degree 5, the Scott-Sorkin algorithm has worst-case running time* $\Omega(r^{19m/100})$.

Proof. The lemma is proven by exhibiting an infinite family of connected instances with maximum degree 5, where only a constant number of vertices have degree less than 5, such that the algorithm may perform $r^{19n/40}$ steps when given an instance on n vertices from this family.

Let n be an integer that is divisible by 40. Let $n_3 = n/5$, $n_4 = 3n/5$, and $n_5 = n/5$. To construct the constraint graph $G_5(n)$, we start with $G_4(n_3 + n_4/2, n_4/2)$ (the construction is given in the proof of Lemma 3.7). Add n_5 new vertices $y_1, y_2, \dots, y_{n_5}$ and edges to form a cycle $(a_{n_4/2+1}, y_1, a_{n_4/2+2}, y_2, \dots, a_{n_4/2+n_3}, y_{n_5}, a_{n_4/2+1})$ (observe that $a_{n_4/2+1}, \cdots, a_{n_4/2+n_3}$ all had degree 3 before adding this cycle). Then, for each vertex y_i, $1 \le i \le n_5$, add three incident edges, connecting y_i to 3 vertices from $\{a_1, \cdots, a_{n_4}\} \cup \{x_2, \cdots, x_{n_4}\}$ in such a way that no vertex has degree greater than 5 and y_1 is incident to a vertex of degree 4. See Figure 4.

On graphs with maximum degree 5, the Scott-Sorkin algorithm performs Reduction 3 on vertices of degree 5, with a preference for those vertices of degree 5 that have neighbors of degree 3 or 4. When there are several choices, we always assume that the algorithm splits on a y vertex with minimum index. Observe that the algorithm splits on $y_1, y_2, \dots, y_{n_5}$, which leaves the graph $G_4(n_3 + n_4/2, n_4/2)$. Thus, by (25), the overall running time is

$$\Omega(r^{n_5} \cdot r^{(n_4/2)/2-2} \cdot r^{(n_3+n_4/2)/4}) = \Omega(r^{n/5+3n/20-2+n/20+3n/40}) = \Omega(r^{19n/40}) = \Omega(r^{19m/100}) \ .$$

This concludes the proof of the lemma. □

It should be noted that tight running time bounds are extremely rare for competitive branching algorithms. Typically, lower bounds proved for branching algorithms are much simpler, using only one branching rule and making sure that this branching rule is applied until the instance has constant size. In the lower bound of this section, however, we managed to make the algorithm branch according to the mixture of branching rules leading to the upper bound — the mixture given by the LP's dual solution. Our message here is twofold. First, when designing lower bounds, it is an advantage to exploit several worst-case branchings of a Measure and Conquer analysis. And second, it could well be that many more existing Measure and Conquer analyses are tight.

4 THE SEPARATE, MEASURE AND CONQUER TECHNIQUE

Our Max $(r, 2)$-CSP algorithm illustrates that one can exploit separator-based branching to design a more efficient exponential-time algorithm. However, the Max $(r, 2)$-CSP problem and its algorithms have certain features that make the analysis simpler than for other problems. In this section, we outline what additional complications could arise in the analysis and how to handle them. In the next section, we will illustrate this more general method. First, the only branching rule of the Max $(r, 2)$-CSP algorithm, Reduction 3, produces r instances with exactly the same constraint graph, and therefore the same measure. As a consequence, the constraints needed to satisfy (2) all become linear. But, typically, the instances produced by a branching rule have different measures, which leads to convex constraints. Second, the measure $\mu_r(R)$ depends only on the number of degree-3 vertices in R. This implies a discretized change in the measure for R whenever L and R are swapped. If the measure attaches incommensurable weights to vertices of different types (different degrees, perhaps), then the change in measure resulting from swapping L and R could take values dense within a continuous domain. Our simple measure for the Max 2-CSP algorithm also implies that the initial separator only needs to balance the *number* of vertices in L and R instead of the *measure* of L and R, which is what is needed more generally. Finally, a general method is needed to combine the separator-based branching, which would typically be done for sparse instances, with the general case, where vertex degrees are arbitrary.

In this section we propose a general method to exploit separator-based branchings in the analysis of an algorithm. The section is tailored to readers familiar with Measure and Conquer and would like to use the present method to design and analyze new algorithms. The will take into account all the complications mentioned in the previous paragraph, and we will illustrate its use in the next section to obtain two faster algorithms for counting dominating sets. The technique will apply to recursive algorithms that label vertices of a graph, and where an instance can be decomposed into two independent subinstances when all the vertices of a separator have been labeled in a certain way. Let $G = (V, E)$ be a graph and $\ell : V \to \mathcal{L}$ be a labeling of its vertices by labels in the finite set $\mathcal{L}$. (Partial labelings are handled by including a label whose interpretation is "unlabeled".) For a subset of vertices $W \subseteq V$, denote by $\mu_r(W)$ and $\mu_s(W)$ two measures for the vertices in W in the graph G labeled by ℓ. The measure μ_r is used for the vertices on the right hand side of the separator and μ_s for the vertices in the separator. Let (L, S, R) be a separation of G. Initially, we use the separation $(L, S, R) = (\emptyset, \emptyset, V)$. We define the measure

$$\mu(L, S, R) = \mu_s(S) + \mu_r(R) + \max\left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right) + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)), \tag{26}$$

where $\epsilon > 0$ is a constant greater than 0 that will be chosen small enough to satisfy constraint (29) below, and B is an arbitrary constant greater than the maximum change (increase or decrease) in imbalance in each transformation in the analysis, except the Separation transformation. The *imbalance* of an instance is $\mu_r(R) - \mu_r(L)$, and we assume, as previously, that

$$\mu_r(R) \geq \mu_r(L). \tag{27}$$

It is important that B be an absolute constant: although the imbalance of an instance may change arbitrarily in an execution of an algorithm, our value of B is only constrained to be greater than the maximum change in imbalance that the analysis takes into account.[3] We need two more assumptions about the measure so that it will be possible to compute a balanced separator. Namely, we will assume that adding a vertex to R (and by symmetry, removing a vertex from R) changes the measure $\mu_r(R)$ by at most a constant. The value of this constant is not crucial for the analysis; so we assume for simplicity that it is at most B (adjusting the value of B if necessary):

$$|\mu_r(R \cup \{v\}) - \mu_r(R)| \leq B \qquad \text{for each } R \subseteq V \text{ and } v \in V. \tag{28}$$

We also assume that $\mu_r(R)$ can be computed in time polynomial in $|V|$ for each $R \subseteq V$.

Let us now look more closely at the measure (26). The terms $\mu_s(S)$ and $\mu_r(R)$ naturally define measures for the vertices in S and R, respectively. No term of the measure directly accounts for the vertices in L; we merely enforce that $\mu_r(R) \geq \mu_r(L)$. The term $\max\left(0, B - \frac{\mu_r(R)-\mu_r(L)}{2}\right)$ is a penalty term based on how balanced the instance is: the more balanced the instance, the larger the penalty term. The penalty term has become continuous, varying from 0 to B. The exact definition of the penalty term will become clearer when we discuss the change in measure when branching. The final term, $(1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S))$, amortizes the increase in measure of at most B due to the balance terms each time the instance is separated.

Let us now formulate some generic constraints that the measure should obey. The first one concerns the separation reduction.

Separation. We assume that an instance with a separation (L, S, R) can be separated into two independent subinstances $(L, S, \emptyset)$ and $(\emptyset, S, R)$ when the labeling of S allows it; specifically, when all vertices in S have been labeled by a subset $\mathcal{L}_s \subseteq \mathcal{L}$ of so-called *separation* labels. The labels are very algorithm-specific. In Max 2-CSP, it sufficed that no label in S was "unlabeled". A Dominating Set algorithm might have $\mathcal{L}_s = \{i, o_d, o_R, o_L\}$, separating the instance when all vertices in the separator have been restricted to be either in the dominating set (i), not in the dominating set and already dominated (o_d), not in the dominating set and needing to be dominated by a vertex in R (o_R), or not in the dominating set and needing to be dominated by a vertex in L (o_L). The separation reduction applies when $\ell(s) \in \mathcal{L}_s$ for each $s \in S$, which arises in two cases. The first is at the beginning of the algorithm when the graph has not been separated, which is represented by the trivial separation $(\emptyset, \emptyset, V)$. The second is when our reductions have produced a separable instance, typically with L and R (and possibly S) nonempty.

Let (L, S, R) be such that $\ell(s) \in \mathcal{L}_s$ for each $s \in S$. The algorithm recursively solves the subinstances $(L, S, \emptyset)$ and $(\emptyset, S, R)$. Let us focus on the instance $(\emptyset, S, R)$; the treatment of the other

[3]Typically, if the imbalance decreases by a very large number for a given branching, it is sufficient to replace this number by a large absolute constant without compromising the quality of the analysis.

instance is symmetric. After a cleanup phase, where the algorithm applies some simplification rules (for example, the vertices labelled o_L and needing to be dominated by a vertex in L can be removed in this subinstance), the next step is to compute a new separator of $S \cup R$. This can be done in various ways, depending on the graph class. For example, polynomial-time computable balanced separators can be derived from upper bounds on the pathwidth of graphs with bounded maximum or average degree [14, 17, 27]. Let us assume that we are dealing with graphs of maximum degree at most 6.

Lemma 4.1. *Let $G = (V, E)$ be a graph of maximum degree at most $\Delta, 3 \le \Delta \le 6$, and μ be a measure as in (26) satisfying (28) such that $\mu_r(R)$ can be computed in time polynomial in $|V|$ for each $R \subseteq V$. A separation (L, S, R) of G can be computed in polynomial time such that $|\mu_r(L) - \mu_r(R)| \le B$ and $|S| \le \alpha_\Delta |V| + o(|V|)$, where $\alpha_3 = 1/6$, $\alpha_4 = 1/3$, $\alpha_5 = 13/30$, and $\alpha_6 = 23/45$.*

Proof. By [17], the pathwidth of G is at most $\alpha_\Delta |V| + o(|V|)$ and a path decomposition of that width can be computed in polynomial time. We view a path decomposition as a sequence of bags $(B_1, \dots, B_b)$ 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 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 [7]. 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 $|\mu_r(L_i) - \mu_r(R_i)| \le B$ by (28). Finding such a bag can clearly be done in polynomial time. □

After a balanced separator (L', S', R') has been computed for $S \cup R$, the instance is solved recursively, and so is the instance $L \cup S$, separated into (L'', S'', R''). Both solutions are then combined into a solution for the instance $L \cup S \cup R$. Without loss of generality, assume $\mu(L', S', R') \ge \mu(L'', S'', R'')$. Assuming that the separation and combination are done in polynomial time, the imposed constraint on the measure is

$$2 \cdot 2^{\mu_r(R')+\mu_s(S')+B+(1+B)\cdot\log_{1+\epsilon}(\mu_r(R')+\mu_s(S'))} \le 2^{\mu_r(R)+\mu_s(S)+(1+B)\cdot\log_{1+\epsilon}(\mu_r(R)+\mu_s(S))}.$$

To satisfy the constraint, it suffices to satisfy the two constraints:

$$\mu_r(R') + \mu_s(S') \le \mu_r(R) + \mu_s(S)$$
$$(1 + B) + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R') + \mu_s(S')) \le (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S))$$

To satisfy both constraints, it is sufficient to constrain that

$$\mu_r(R) + \mu_s(S) \ge (1 + \epsilon)(\mu_r(R') + \mu_s(S')). \tag{29}$$

This is the only constraint involving the size of a separator. It constrains that separating $(\emptyset, S, R)$ to (L', S', R') should reduce $\mu_r(R) + \mu_s(S)$ by a constant factor, namely $1 + \epsilon$.

Branching. The branching rules will apply to two kinds of instances: balanced and imbalanced instances. We say that an instance is *balanced* if $0 \le \mu_r(R) - \mu_r(L) \le 2B$ and *imbalanced* if $\mu_r(R) - \mu_r(L) > 2B$. For a balanced instance, the measure (26) is

$$\mu(L, S, R) = \mu_s(S) + \frac{\mu_r(R)}{2} + \frac{\mu_r(L)}{2} + B + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S))$$

and for an imbalanced instance, it is

$$\mu(L, S, R) = \mu_s(S) + \mu_r(R) + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)).$$

A standard Measure and Conquer analysis would probably assign a weight to every vertex and use a measure of the form $\mu_r(L) + \mu_s(S) + \mu_r(R)$, while our measure does not assign weights to vertices in L. Let us recall the optimistic assumptions in Section 1.1 where we outlined an analysis for Max (2, 2)-CSP under ideal conditions: after exhausting the separator S, we assumed half of the other vertices that were removed by the branching were in L and the other half in R. Thus the instance decomposed into two equally-small connected components. We will now present a condition under which $\mu_r(R)$ decreases at least by half as much as $\mu_r(L) + \mu_r(R)$.

Suppose a transformation taking (L, S, R, ℓ) to (L', S', R', ℓ') decreases $\mu_r(R) + \mu_r(L)$ by d. Since the measure includes roughly (and at least) half of $\mu_r(R) + \mu_r(L)$, in an ideal case the measure $\mu_r(R) + \max\left(0, B - \frac{\mu_r(R)-\mu_r(L)}{2}\right)$ decreases by $d/2$. We will now show that this is indeed the case for our measure if the transformation guarantees the following condition:

$$\text{If } \mu_r(R) - \mu_r(L) > B, \text{ then } \mu_r(R) - \mu_r(R') \geq \mu_r(L) - \mu_r(L') \quad . \tag{30}$$

Recall that B was defined to be greater than the change in imbalance in the analysis of each transformation, so $\mu_r(R) - \mu_r(L) \leq B$ implies the instance remains balanced after a transformation. Condition (30) is very natural, expressing that, if the instance is imbalanced or risks becoming imbalanced we would like to make more progress on the large side.

In the balanced case, $0 \leq \mu_r(R) - \mu_r(L) \leq 2B$. If $\mu_r(R) - \mu_r(L) > B$, then by (30), $\mu_r(R') - \mu_r(L') \leq \mu_r(R) - \mu_r(L) \leq 2B$, while if $\mu_r(R) - \mu_r(L) \leq B$, then by the definition of B, $\mu_r(R') - \mu_r(L') \leq 2B$. It follows that the resulting instance (L', S', R', ℓ') is balanced as well, and $\mu_r(R) + \max\left(0, B - \frac{\mu_r(R)-\mu_r(L)}{2}\right)$ decreases by $(\mu_r(R) + \mu_r(L) - \mu_r(R') - \mu_r(L'))/2 = d/2$.

In the imbalanced case, $\mu_r(R) - \mu_r(L) > 2B$. By condition (30), $\mu_r(R) + \max\left(0, B - \frac{\mu_r(R)-\mu_r(L)}{2}\right)$ decreases by

$$\begin{aligned}\mu_r(R) - \mu_r(R') &\geq (\mu_r(R) - \mu_r(R') + \mu_r(L) - \mu_r(L'))/2 \\ &= d/2\end{aligned}$$

if (L', S', R', ℓ') is also imbalanced, or by

$$\begin{aligned}\mu_r(R) - \left(\frac{\mu_r(R')}{2} + \frac{\mu_r(L')}{2} + B\right) &= \frac{\mu_r(R)}{2} - \left(B - \frac{\mu_r(R)}{2}\right) - \frac{\mu_r(R')}{2} - \frac{\mu_r(L')}{2} \\ &\geq (\mu_r(R) + \mu_r(L) - \mu_r(R') - \mu_r(L'))/2 \\ &= d/2\end{aligned}$$

if (L', S', R', ℓ') is balanced.

Thus, if Condition (30) holds, i.e., we can guarantee more progress on the large side if the instance is imbalanced or risks to become imbalanced, then the analysis is at least as good as a non-separator based analysis, but with the additional improvement due to the separator branching.

Integration into a standard Measure and Conquer analysis. The Separate, Measure and Conquer analysis will typically be used when the instance has become sufficiently sparse that one can guarantee that a small separator exists. We can view the part played by the Separate, Measure and Conquer analysis as a subroutine and integrate it into any other Measure and Conquer analysi. We only need guarantee that the measure of an instance does not increase when transitioning to the subroutine. Formally, this is done with the following lemma that can be proved with a simple induction.

Lemma 4.2 ([27, 30]). *Let A be an algorithm for a problem P, B be an algorithm for (special instances of) P, $c \geq 0$ and $r > 1$ be constants, and $\mu(\cdot), \mu'(\cdot), \eta(\cdot)$ be measures for the instances of P, such that for any input instance I, $\mu'(I) \leq \mu(I)$, and for any input instance I, A either solves P on I by invoking B with running time $O(\eta(I)^{c+1})r^{\mu'(I)}$, or reduces I to 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),*

$$(\forall i) \quad \eta(I_i) \leq \eta(I) - 1, \text{ and} \tag{31}$$

$$\sum_{i=1}^{k} r^{\mu(I_i)} \leq r^{\mu(I)}. \tag{32}$$

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

This completes the description of the Separate, Measure and Conquer method. We now illustrate its use to obtain improved algorithms for #Dominating Set.

5 COUNTING DOMINATING SETS

Dominating Set (DS) and its variants are of central importance in exponential-time algorithms. Let us denote the number of vertices of the input graph by n. DS can be solved in time $O(1.4864^n)$ using polynomial space [38] and in time $O(1.4689^n)$ using exponential space [38]. All minimal dominating sets can be enumerated in time $O(1.7159^n)$ and polynomial space [21]. Minimum Weighted Dominating Set can be solved in time $O(1.5535^n)$ and exponential space [17]. Partial Dominating Set can be solved in time $O(1.5673^n)$ using polynomial space [52] and in time $O(1.5012^n)$ using exponential space [52]. Other variants have been considered (see, e.g., [45, 61]) and the problems have been studied on many graph classes (see, e.g., [10, 11, 28, 34, 43, 46, 53]). The #DS problem is to compute, for a given graph G, the function $d : \{0, \ldots, n\} \to \mathbb{N}$ such that for each $k \in \{0, \ldots, n\}$, $d(k)$ is the number of dominating sets of G of size k.

The current fastest polynomial-space algorithm for #DS is a $O(1.5673^n)$ time algorithm by van Rooij [60].[4] Like many DS algorithms, it uses a construction due to Grandoni [37] to transform the instance into an instance for Set Cover. Given a multiset $\mathcal{S}$ of subsets of a universe $\mathcal{U}$, a collection of subsets $C \subseteq \mathcal{S}$ is a *set cover* if $\bigcup_{c \in C} c = \mathcal{U}$. The transformation from DS to Set Cover sets $\mathcal{U} := V$ and adds the set $N_G[v]$ to $\mathcal{S}$ for each vertex v. Here, $N_G[v]$ denotes the closed neighborhood of v, that is, the set of vertices containing v and all its neighbors. While many algorithms for DS and variants rely on this transformation to Set Cover, the current fastest polynomial-space algorithm for subcubic graphs is a simple $O^\star\left(2^{n/2}\right)$ time algorithm [41] that works directly on the input graph. (It still outperforms all more recent algorithms on general graphs [18, 38, 62] when their analysis is restricted to the subcubic case.)

In this section, we apply the Separate, Measure and Conquer method to design and analyze faster algorithms for #DS for subcubic graphs and, separately, for general graphs. For subcubic graphs we design an algorithm working directly on the input graph, where we can essentially reuse the Max (3, 2)-CSP analysis of Section 3. For general graphs, our algorithm is based on the Set Cover translation and essentially just adds separation to [60].

[4] Using exponential space, the problem can be solved in time $O(1.5002^n)$ [53].

5.1 Subcubic graphs

We assume the input graph G has maximum degree at most 3, and we denote by $\Gamma(G)$ the *cubic structure* of G, the unique 3-regular graph obtained from G by exhaustively contracting edges incident to vertices of degree at most 2 and removing isolated vertices. Our algorithm will branch on the vertices of $\Gamma(G)$ in the same way as the Max $(r, 2)$-CSP algorithm and label the vertices of G so as to remember which vertices still need to be dominated, which ones should not be dominated, etc. Once the number of vertices in $\Gamma(G)$ is upper bounded by a constant, the problem can be solved in polynomial time by tree-decomposition methods. We will first describe the labeling, then the base case where the number of vertices of $\Gamma(G)$ is bounded by a constant, and then the branching strategy.

To solve the #DS problem, the algorithm applies labels U, N, and C to vertices. We denote by $\ell(v) \in \{U, N, C\}$ the label of vertex v. The algorithm counts the number of sets D for each size k, $0 \le k \le |V|$, with the following restrictions according to the label of each vertex v:

- "unlabeled" U: v needs to be dominated by D;
- "not in" N: v is not in D, but needs to be dominated by D;
- "covered" C: there is no restriction on v, i.e., v may or may not be in D and v does not need to be dominated by D.

Initially, all vertices are labeled U. There is no label for vertices that do not need to be dominated and do not belong to a dominating set, since they are simply deleted from the graph. Vertices that are added to the dominating set are also removed from the graph and their neighbors are updated to reflect that they do not need to be dominated any more: neighbors labeled U or C get label C, and neighbors labeled N are deleted. When we speak of a dominating set of a labeled graph G, we mean a vertex set containing vertices labeled U and C that dominates all vertices labeled U and N.

If $|V(\Gamma(G))| \le A$ for a large enough constant $A \ge 2$, the instance is solved in polynomial time as follows. First, a tree decomposition of G of width at most A is computed. To do this, we start from a trivial tree decomposition of $\Gamma(A)$ that has just one bag with all the vertices of $\Gamma(A)$. In polynomial time, one can augment that tree decomposition with the vertices from $V(G) \setminus V(\Gamma(A))$ and make sure that its width does not exceed A [6]. Intuitively, each degree-0 vertex can get its own private bag and if a vertex v of degree at most 2 was contracted away, choose a bag containing its neighbor(s) and add a neighboring bag containing v and its neighbor(s). Now, use a tree decomposition based algorithm, such as [63], to solve the instance in polynomial time. The latter treats unlabeled graphs but needs only minor adaptations for labeled graphs.

We will now describe the algorithm for the case when $|V(\Gamma(G))| > A$. The algorithm will satisfy the invariant that degree-3 vertices in G are labeled U. It will compute an $(n + 1)$-size vector $\#\mathrm{ds}_G$ such that the labeled graph G has $\#\mathrm{ds}_G[i]$ dominating sets of size i, $0 \le i \le n$.

Analogous to the algorithm in Section 3, we now define a Reduction 3. (Analogs of Reduction 0–2 are not needed here.)

Reduction 3 Let x be a vertex of degree 3. Thus, $\ell(x) = U$. There are three subinstances, $G_{\text{in}}, G_{\text{opt}}$, and G_{forb} (thought of as "in", "optional", and "forbidden"). They are obtained from G as follows.

- In G_{in}, we add x to the dominating set. The vertex x is deleted from the graph, its neighbors labeled U are relabeled C, and its neighbors labeled N are deleted.
- In G_{opt}, we prevent x from being in the dominating sets and we remove the requirement that it needs to be dominated. The vertex x is removed from the graph.
- In G_{forb}, we forbid that x is dominated, that is we are interested in the number of dominating sets of $G - x$ that do not contain any vertex from $N_G(x)$. Delete the vertices in $N_G(x)$ labeled C, assign label N to the remaining vertices of $N_G(x)$, and delete x.

The algorithm computes the number of dominating sets of size i of G by setting

$$\#\mathrm{ds}(G)[i] = \#\mathrm{ds}(G_{\mathrm{in}})[i-1] + \#\mathrm{ds}(G_{\mathrm{opt}})[i] - \#\mathrm{ds}(G_{\mathrm{forb}})[i].$$

The algorithm returns the vector #ds.

To see that Reduction 3 correctly computes the number of dominating sets of size i of G, observe that these sets that contain the vertex x are counted in $\#\mathrm{ds}(G_{\mathrm{in}})[i-1]$. The dominating sets of size i of G that do not contain x are the dominating sets of size i of $G - x$, counted in $\#\mathrm{ds}(G_{\mathrm{opt}})[i]$, except for those that do not dominate x (counted in $-\#\mathrm{ds}(G_{\mathrm{forb}})[i]$).

We observe that Reduction 3 has at least the same effect on $\Gamma(G)$ as Reduction 3 in Max 2-CSP algorithm of Section 3 has on the constraint graph: the vertex x is deleted, and all vertices with degree at most 2 in $\Gamma(G - x)$ are contracted or, even better, deleted. Reductions 0-2 are replaced by operations doing nothing. This has the same effect on $\Gamma(G)$ as Reductions 0-2 in Max 2-CSP algorithm of Section 3 has on the constraint graph, which either becomes cubic again or vanishes. Thus, the algorithm will select the pivot vertex for branching in exactly the same way as the algorithm in Section 3. We note that the running time analysis of Section 3 was based only on the measure of the constraint graph. Since degree-3 vertices are handled by a 3-way branching, the running time analysis for Max $(3, 2)$-CSP applies for #DS as well, giving a running time of $3^{n/5+o(n)} = O(1.2458^n)$ for subcubic graphs. This improves on the previously fastest polynomial-space algorithm with running time $O^\star\left(2^{n/2}\right) = O(1.4143^n)$ [41].

Theorem 5.1. *The described algorithm solves #DS in $3^{n/5+o(n)}$ time and polynomial space on subcubic graphs.*

There are two novel ideas that make this improved running time possible, and we will briefly discuss how much each idea contributed to the improvement. The first one is branching on vertices in a small separator, and the second one is our new 3-way branching. Our 3-way branching is inspired by the inclusion/exclusion branching of [64], but to the best of our knowledge, it has not been used in this form before. The algorithm of [41] used a 4-way branching that gave a running time of $O^\star\left(4^{n/4}\right) = O^\star\left(2^{n/2}\right) = O(1.4143^n)$. If we use their analysis, or the analysis of [57], together with our 3-way branching, one obtains a running time bound of $O^\star\left(3^{n/4}\right) = O(1.3161^n)$. The further improvement to $3^{n/5+o(n)} = O(1.2458^n)$ is due solely to the separator branching.

5.2 General graphs

As another application of our Separate, Measure and Conquer technique, we show that by changing the preference for the pivot vertex in the subcubic subproblem, taking advantage of a separator, the running time of van Rooij's algorithm [60] improves from $O(1.5673^n)$ to $O(1.5183^n)$.

Let us recall van Rooij's algorithm #SC-vR [60] (see Algorithm 1). It first reduces the #DS problem to the #Set Cover problem, where each vertex corresponds to an element of the universe $\mathcal{U}$ and the closed neighborhood of each vertex corresponds to a set $X \in \mathcal{S}$. Then, there is a cardinality-preserving bijection between dominating sets of the graph and set covers of the instance $(\mathcal{U}, \mathcal{S})$. The *incidence graph* of $(\mathcal{U}, \mathcal{S})$ is the bipartite graph $(\mathcal{S} \cup \mathcal{U}, E)$ which has an edge $(X, e) \in E$ between a set $X \in \mathcal{S}$ and an element $e \in \mathcal{U}$ if and only if $e \in X$. In the #Set Cover problem, the input is an incidence graph of an instance $(\mathcal{U}, \mathcal{S})$, and the task is to compute for each size κ, $0 \le \kappa \le |\mathcal{S}|$, the number of set covers of size κ. As an additional input, the algorithm has a set A of annotated vertices that is initially empty. When the algorithm finds a vertex of degree at most 1 or a vertex of degree 2 that has the same neighborhood as some other vertex, it annotates this vertex, which means the vertex is effectively removed from the instance, and is left to a polynomial-time dynamic programming algorithm, #SC-DP, that handles instances of maximum degree at most 2 plus annotated vertices.

ALGORITHM 1: #SC-vR(I, A) – van Rooij's algorithm for #Set Cover.

Input: The incidence graph $I = (\mathcal{S} \cup \mathcal{U}, E)$ of $(\mathcal{U}, \mathcal{S})$ and a set of annotated vertices A
Output: A list containing the number of set covers of $\mathcal{S}$ of each cardinality κ

if *there exists a vertex* $v \in V(I) \setminus A$ *of degree at most one in* $I - A$ **then**
  return #SC-vR($I, A \cup \{v\}$)
else if *there exist two vertices* $v_1, v_2 \in V(I) \setminus A$ *both of degree two in* $I - A$ *that have the same two neighbors* **then**
  return #SC-vR($I, A \cup \{v_1\}$)
else
  Let $X \in \mathcal{S} \setminus A$ be a set vertex of maximum degree in $I - A$
  Let $e \in \mathcal{U} \setminus A$ be an element vertex of maximum degree in $I - A$
  if $d_{I-A}(X) \le 2$ ***and*** $d_{I-A}(e) \le 2$ **then**
    return #SC-DP(I)
  else if $d_{I-A}(X) > d_{I-A}(e)$ **then**
    Let L_{take} = #SC-vR($I - N_I[X], A \setminus N_I(X)$) and increase all cardinalities by one
    Let L_{discard} = #SC-vR($I - X, A$)
    return $L_{\text{take}} + L_{\text{discard}}$
  else
    Let L_{optional} = #SC-vR($I - e, A$)
    Let $L_{\text{forbidden}}$ = #SC-vR($I - N_I[e], A \setminus N_I(e)$)
    return $L_{\text{optional}} - L_{\text{forbidden}}$

Van Rooij showed that this algorithm has running time $O(1.5673^n)$. We will now modify it to obtain Algorithm #SC (see Algorithm 2), which takes advantage of small separators in subcubic graphs. Then, we will show that this modification improves the running time to $O(1.5183^n)$. Algorithmically, the change is simple. The parts that remain essentially unchanged (except that the separation is passed along in the recursive calls) are written in gray in Algorithm 2. For convenience, we also define as a function what it means to branch on a vertex (see Function Branch) since we use it several times. When the algorithm reaches an instance with maximum degree at most 3, it computes a separation of the incidence graph, minus the annotated vertices, and prefers to branch on vertices in the separator. To do this, we add to the input a separation (L, S, R) of $I - A$. Initially, this separation is $(\emptyset, \emptyset, V(I) - A)$; it is really only used when the maximum degree of $I - A$ decreases to 3. We note that a separation (L, S, R) for $I - A$ can easily be transformed into a separation (L', S', R') for I. For this, we assign each vertex from A to one of its neighbors in $V(I) \setminus A$ when it is annotated (or to an arbitrary vertex if it has degree 0). Now, start with $(L', S', R') = (L, S, R)$. A vertex from A is added to L', S', or R' if it is assigned to a vertex that is in L', S', or R', respectively. The vertices in A will not affect the measure, defined momentarily. Our algorithm also explicitly handles connected components when the graph is not connected, and it calls a subroutine, #3SC (Algorithm 3), when the maximum degree is 3.

We are now ready to upper bound the running time of Algorithm #SC (Algorithm 2). For instances with maximum degree at least 4, we use the same measure as van Rooij,

$$\mu_4 = \sum_{e \in \mathcal{U} \setminus A} w_{\text{elt}}(d_{I-A}(e)) + \sum_{X \in \mathcal{S} \setminus A} w_{\text{set}}(d_{I-A}(X)) \ .$$

ALGORITHM 2: #SC$(I, A, (L, S, R))$ – separator-based #Set Cover algorithm. (Lines shown in grey are essentially unchanged from Algorithm 1.)

Input: The incidence graph $I = (\mathcal{S} \cup \mathcal{U}, E)$ of $(\mathcal{U}, \mathcal{S})$, a set of annotated vertices A, and a separation (L, S, R) of $I - A$
Output: A list containing the number of set covers of $\mathcal{S}$ of each cardinality κ

if *I has a connected component $X \subsetneq V(I)$* **then**
  Let $L_X = \#\mathrm{SC}(I[X], A \cap X, (\emptyset, \emptyset, X \setminus A))$
  Let $L_{\overline{X}} = \#\mathrm{SC}(I - X, A \setminus X, (\emptyset, \emptyset, V(I) \setminus (X \cup A))$
  return L where $L[i] = \sum_{j=0}^{i} L_X[j] \cdot L_{\overline{X}}[i-j]$
else if *there exists a vertex $v \in V(I) \setminus A$ of degree at most one in $I - A$* **then**
  return $\#\mathrm{SC}(I, A \cup \{v\}, (L \setminus \{v\}, S \setminus \{v\}, R \setminus \{v\}))$
else if *there exist two vertices $v_1, v_2 \in V(I) \setminus A$ both of degree two in $I - A$ that have the same two neighbors* **then**
  return $\#\mathrm{SC}(I, A \cup \{v_1\}, (L \setminus \{v_1\}, S \setminus \{v_1\}, R \setminus \{v_1\}))$
else
  Let $X \in \mathcal{S} \setminus A$ and $e \in \mathcal{U} \setminus A$ be a set vertex and an element vertex, each of maximum degree in $I - A$
  if $d_{I-A}(X) \leq 2$ ***and*** $d_{I-A}(e) \leq 2$ **then**
    return #SC-DP(I)
  else if $d_{I-A}(X) \leq 3$ ***and*** $d_{I-A}(e) \leq 3$ **then**
    return #3SC$(I, A, (L, S, R))$
  else if $d_{I-A}(X) > d_{I-A}(e)$ **then**
    return Branch$(I, A, (L, S, R), X)$
  else
    return Branch$(I, A, (L, S, R), e)$

Function Branch$(I, A, (L, S, R), v)$

Input: The incidence graph $I = (\mathcal{S} \cup \mathcal{U}, E)$ of $(\mathcal{U}, \mathcal{S})$, a set of annotated vertices A, and a separation (L, S, R) of $I - A$, and a vertex $v \in (\mathcal{S} \cup \mathcal{U}) \setminus A$ to branch on
Output: A list containing the number of set covers of $\mathcal{S}$ of each cardinality κ

if $v \in \mathcal{S}$ **then**
  Let $L_{\text{discard}} = \#\mathrm{SC}(I - v, A, (L \setminus \{v\}, S \setminus \{v\}, R \setminus \{v\}))$
  Let $L_{\text{take}} = \#\mathrm{SC}(I - N_I[v], A \setminus N_I[v], (L \setminus N[v], S \setminus N[v], R \setminus N[v]))$ and increase all cardinalities by one
  return $L_{\text{take}} + L_{\text{discard}}$
else
  Let $L_{\text{optional}} = \#\mathrm{SC}(I - v, A, (L \setminus \{v\}, S \setminus \{v\}, R \setminus \{v\}))$
  Let $L_{\text{forbidden}} = \#\mathrm{SC}(I - N_I[v], A \setminus N_I[v], (L \setminus N[v], S \setminus N[v], R \setminus N[v]))$
  return $L_{\text{optional}} - L_{\text{forbidden}}$

Here, $w_{\text{elt}}(\cdot)$ and $w_{\text{set}}(\cdot)$ are non-negative real functions of the vertex degrees, which will be determined later. For instances with maximum degree 3, we use the following measure,

$$\mu_3 = \mu_s(S) + \mu_r(R) + \max\left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right) + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)),$$

ALGORITHM 3: #3SC$(I, A, (L, S, R))$ – separator-based #Set Cover algorithm for subcubic instances.

Input: The incidence graph $I = (\mathcal{S} \cup \mathcal{U}, E)$ of $(\mathcal{U}, \mathcal{S})$, a set of annotated vertices A, and a separation (L, S, R) of $I - A$

Output: A list containing the number of set covers of $\mathcal{S}$ of each cardinality κ

if $S = \emptyset$ **then**
  Compute a balanced separation (L, S, R) of $I - A$ with respect to the measure μ_r using Lemma 4.1

if $\mu_r(L) > \mu_r(R)$ **then**
  Swap L and R

if *there exists a vertex* $s \in S$ *with no neighbor in* L **then**
  return #SC$(I, A, (L, S \setminus \{s\}, R \cup \{s\}))$

else if *there exists a vertex* $s \in S$ *with no neighbor in* R **then**
  return #SC$(I, A, (L \cup \{s\}, S \setminus \{s\}, R))$

else if *there exists a vertex* $s \in S$ *with* $d_{I-A}(s) = 2$ **then**
  if s *has a degree-2 neighbor in* $I - A$ **then**
    Let r be the first degree-3 vertex or vertex from S encountered when moving from s to the right along a path of degree-2 vertices in $I - A$
    return Branch$(I, A, (L, S, R), r)$
  else if (L, S, R) *is balanced* **then**
    return #SC$(I, A, (L \setminus \{l\}, (S \setminus \{s\}) \cup \{l\}, R \cup \{s\}))$ where $\{l\} = L \cap N_{I-A}(s)$
  else
    return #SC$(I, A, (L \cup \{s\}, (S \setminus \{s\}) \cup \{r\}, R \setminus \{r\}))$ where $\{r\} = R \cap N_{I-A}(s)$

else if $\mu_r(R) - \mu_r(L) > B$ *and there exists a vertex* $s \in S$ *with two neighbors in* L *and one neighbor* r *in* R *that has degree* 3 *in* $I - A$ **then**
  return #SC$(I, A, (L \cup \{s\}, (S \cup \{r\}) \setminus \{s\}, R \setminus \{r\}))$

else if $\mu_r(R) - \mu_r(L) > B$ *and there exists a vertex* $s \in S$ *with two neighbors in* L *and one neighbor* r *in* R *with* $N_{I-A}(r) = \{s, s'\}$ *for some* $s' \in S$ **then**
  return #SC$(I, A, (L \cup \{s, r\}, S \setminus \{s\}, R \setminus \{r\}))$

else if *there exists an element* $s \in \mathcal{U} \cap S$ **then**
  return Branch$(I, A, (L, S, R), s)$

else
  Let $s \in \mathcal{S} \cap S$ be a set in S
  return Branch$(I, A, (L, S, R), s)$

where

$$\mu_s(S) = \sum_{s \in S} w_{\text{sep}}(d_{I-A}(s)),$$
$$\mu_r(R) = \sum_{r \in R} w_{\text{right}}(d_{I-A}(r)), \text{ and}$$
$$B = 6 \cdot w_{\text{right}}(3).$$

Again, $w_{\text{sep}}(\cdot)$ and $w_{\text{right}}(\cdot)$ are non-negative real functions of the vertex degrees, which will be determined later. We commonly refer to $w_{\text{elt}}(i)$, $w_{\text{set}}(i)$, $w_{\text{sep}}(i)$, $w_{\text{right}}(i)$ as the weight of an element vertex, a set vertex, a separator vertex and a right vertex, respectively, of degree i.

To combine the analysis for subcubic instances and instances with maximum degree at least 4 using Lemma 4.2, we impose the following constraints, guaranteeing that $\mu_4 \geq \mu_3$ when $\mu_4 = \omega(1)$. (When $\mu_4 = O(1)$, the algorithm will take constant time.)

$$w_{\text{elt}}(3) > w_{\text{right}}(3) \qquad w_{\text{set}}(3) > w_{\text{right}}(3) \tag{33}$$

$$w_{\text{elt}}(i) \geq w_{\text{right}}(i), \text{ and} \qquad w_{\text{set}}(i) \geq w_{\text{right}}(i), \qquad i \in \{0, 1, 2\}. \tag{34}$$

For instances with degree at least 4, we obtain exactly the same constraints on the measure as in [60]. Denoting $\Delta w_{\text{elt}}(i) = w_{\text{elt}}(i) - w_{\text{elt}}(i-1)$ and $\Delta w_{\text{set}}(i) = w_{\text{set}}(i) - w_{\text{set}}(i-1)$, these constraints are

$$w_{\text{elt}}(0) = w_{\text{elt}}(1) = 0, \qquad w_{\text{set}}(0) = w_{\text{set}}(1) = 0, \tag{35}$$

$$\Delta w_{\text{elt}}(i) \geq 0, \qquad \Delta w_{\text{set}}(i) \geq 0 \qquad \text{for all } i \geq 2, \tag{36}$$

$$\Delta w_{\text{elt}}(i) \geq \Delta w_{\text{elt}}(i+1), \qquad \Delta w_{\text{set}}(i) \geq \Delta w_{\text{set}}(i+1) \qquad \text{for all } i \geq 2, \tag{37}$$

$$2 \cdot \Delta w_{\text{elt}}(3) \leq w_{\text{elt}}(2), \text{ and} \qquad 2 \cdot \Delta w_{\text{set}}(4) \leq w_{\text{set}}(2). \tag{38}$$

For branching on a vertex with degree $d \geq 4$ with r_i neighbors of degree i in $I-A$, where $\sum_{i=2}^{d} r_i = d$, we have the following constraints,

$$2^{-w_{\text{set}}(d)-\sum_{i=2}^{\infty} r_i \cdot w_{\text{elt}}(i) - \Delta w_{\text{set}}(d)\cdot\sum_{i=2}^{\infty} r_i\cdot(i-1)} + 2^{-w_{\text{set}}(d)-\sum_{i=2}^{\infty} r_i \cdot \Delta w_{\text{elt}}(i)} \leq 1 \tag{39}$$

if the vertex is a set vertex, and

$$2^{-w_{\text{elt}}(d)-\sum_{i=2}^{\infty} r_i \cdot w_{\text{set}}(i) - \Delta w_{\text{elt}}(d)\cdot\sum_{i=2}^{\infty} r_i\cdot(i-1)} + 2^{-w_{\text{elt}}(d)-\sum_{i=2}^{\infty} r_i \cdot \Delta w_{\text{set}}(i)} \leq 1 \tag{40}$$

if it is an element vertex. We note that the constraints (39) have $r_d = 0$ since the algorithm prefers to branch on elements when there is both an element and a set of maximum degree at least 4. Constraints (35)–(40) are directly taken from [60].

For instances where $I - A$ has maximum degree 3, let us recall (from Section 4) that an instance is *balanced* if $0 \leq \mu_r(R) - \mu_r(L) \leq 2B$ and *imbalanced* if $\mu_r(R) - \mu_r(L) > 2B$. For convenience, we define the minimum decrease in measure due to a decrease in the degree in $I - A$ of a vertex from $S \cup R$ of degree 2 or 3 in a balanced or imbalanced instance:

$$\delta_{\text{deg-dec}} = \min_{i \in \{2,3\}} \left\{ w_{\text{sep}}(i) - w_{\text{sep}}(i-1), \frac{w_{\text{right}}(i) - w_{\text{right}}(i-1)}{2} \right\}. \tag{41}$$

To make sure that annotating vertices does not increase the measure, we require that the decrease of the degree of a vertex (half-edge deletion) does not increase the measure:

$$\delta_{\text{deg-dec}} \geq 0. \tag{42}$$

For computing a new separator in line 2, constraint (29) becomes

$$\begin{aligned} w_{\text{sep}}(3)/6 + 5/12 \cdot w_{\text{right}}(3) &< w_{\text{right}}(3), \text{ or} \\ w_{\text{sep}}(3) &< 7/2 \cdot w_{\text{right}}(3). \end{aligned} \tag{43}$$

For dragging a separator vertex into R if it has no neighbor in L (line 6), imbalanced instances are most constraining:

$$-w_{\text{sep}}(d) + w_{\text{right}}(d) \leq 0, \qquad 2 \leq d \leq 3. \tag{44}$$

For dragging a separator vertex into L if it has no neighbor in R (line 8), balanced instances are most constraining, imposing the constraints

$$-w_{\text{sep}}(d) + 1/2 \cdot w_{\text{right}}(d) \leq 0, \qquad 2 \leq d \leq 3,$$

which are no more constraining than (44).

If there is a vertex $s \in S$ with $d_{I-A}(s) = 2$ (line 9), it has one neighbor in $L \setminus A$ and one neighbor in $R \setminus A$, otherwise a previous case would have applied. The algorithm distinguishes three cases. In the first case, s has a neighbor $x \in L \cup R$ that has degree 2 in $I - A$. The algorithm branches on r, which is the first vertex we encounter when moving from s to the right in $I - A$, that is not a degree-2 vertex in R. So, r is either in S or it is a degree-3 vertex in R. If $r \in S$, then branching on r removes the vertex r from the instance, and the vertex s is also removed from $I - A$, either directly, or by triggering the degree-1 rule in line 6 of Algorithm #SC. Thus, the measure decreases by at least $2 \cdot w_{\text{sep}}(2)$ in both branches. Thus, we will constrain that

$$2 \cdot 2^{-2 \cdot w_{\text{sep}}(2)} \leq 1$$

or, equivalently, that

$$2 \cdot w_{\text{sep}}(2) \geq 1. \tag{45}$$

We could weaken the constraint further by analyzing the effect on the measure of the other neighbors of s and r, but the constraint will not turn out to be tight. If $r \notin S$, then $r \in R$ and r has degree 3 in $I - A$. After branching on r, we have a measure decrease of at least $w_{\text{right}}(3)/2$ because r is removed (r contributes $w_{\text{right}}(3)$ to the measure in the imbalanced case but only $w_{\text{right}}(3)/2$ in the balanced case). In addition, we have a measure decrease of $w_{\text{sep}}(2)$ because s is removed, a decrease of $w_{\text{right}}(2)/2$ because x is removed, and a decrease of at least $2 \cdot \delta_{\text{deg-dec}}$ because two more neighbors of r have their degree reduced or are deleted. We note that x could be in L and its removal decreases $\mu_r(L)$, but we account for a decrease of $\mu_r(R)$ that is at least as large since $w_{\text{right}}(3) \geq w_{\text{right}}(2)$ due to constraints (41) and (42). Since Condition (30) holds, the decrease of μ due to vertices in $R \cup L$ is therefore at least half the decrease of $\mu_r(L) + \mu_r(R)$. For this subcase, we constrain that

$$2 \cdot 2^{-w_{\text{sep}}(2) - 2 \cdot \delta_{\text{deg-dec}} - (w_{\text{right}}(2) + w_{\text{right}}(3))/2} \leq 1$$

or, equivalently, that

$$w_{\text{sep}}(2) + 2 \cdot \delta_{\text{deg-dec}} + (w_{\text{right}}(2) + w_{\text{right}}(3))/2 \geq 1. \tag{46}$$

In the second case, s has two neighbors with degree 3 and the instance is balanced. The algorithm "drags" s to the right, that is, s is moved from S to R and its neighbor in L is moved to S. We obtain the constraint

$$-w_{\text{sep}}(2) + w_{\text{sep}}(3) + 1/2 \cdot (w_{\text{right}}(2) - w_{\text{right}}(3)) \leq 0. \tag{47}$$

In the third case, s has two neighbors with degree 3 and the instance is imbalanced. The algorithm drags s to the left, which leads to the weaker constraint

$$-w_{\text{sep}}(2) + w_{\text{sep}}(3) - w_{\text{right}}(3) \leq 0.$$

For the operation in line 18, where $\mu_r(R) - \mu_r(L) > B$ and some vertex $s \in S$ has two neighbors in L and one degree-3 neighbor in R, the vertex s is dragged to the left. In the worst case, the resulting instance is balanced, imposing the following constraints on the measure,

$$-w_{\text{sep}}(3) + w_{\text{sep}}(3) + 1/2 \cdot (w_{\text{right}}(3) - w_{\text{right}}(3)) \leq 0,$$

which always holds.

For the operation in line 20, where $\mu_r(R) - \mu_r(L) > B$ and some vertex $s \in S$ has two neighbors in L and one degree-2 neighbor r in R that has another neighbor s' in S, the vertices s and r are dragged to the left. In the worst case, the resulting instance is balanced, imposing the following constraints on the measure,

$$-w_{\text{sep}}(3) + 1/2 \cdot w_{\text{right}}(3) \le 0,$$

which holds due to (44).

For the constraints of the branching steps, we will take into account the decrease of vertex degrees in the second neighborhood of a set that we add to the set cover and of an element that we forbid to cover. The analysis will be simplified by imposing the following constraints,

$$2 \cdot \Delta w_{\text{sep}}(3) \le w_{\text{sep}}(2) \qquad 2 \cdot \Delta w_{\text{right}}(3) \le w_{\text{right}}(2). \tag{48}$$

They will enable us to account for a decrease in measure of $\delta_{\text{deg-dec}}$ for each edge that leaves the second neighborhood, even when two of these edges are incident to one vertex of degree 2. We cannot relax these equations for sets as in (38): When $I - A$ has maximum degree 3 and it contains an element of degree 3, then van Rooij's algorithm prefers to branch on a degree-3 element, whereas the new algorithm cannot guarantee to find a degree-3 element in S.

For the two branching rules (lines 21–8 and 24–25), the algorithm selects a separator vertex $s \in S$, which has degree 3 in $I - A$ due to line 9. In the first branch, s is deleted from the graph, and in the second branch, $N_I[s]$ is deleted.

First, consider the case where s has a neighbor s' in S. By lines 5 and 7, $N_{I-A}(s) = \{l, s', r\}$ and $N_{I-A}(s') = \{l', s, r'\}$ with $\{l, l'\} \subseteq L$ and $\{r, r'\} \subseteq R$. Since I is bipartite, we have that $l \ne l'$ and $r \ne r'$. In the balanced case, the branch where only s is deleted has a decrease in measure of $w_{\text{sep}}(3)$ for s, $\Delta w_{\text{sep}}(3)$ for s', $1/2 \cdot \Delta w_{\text{right}}(d_{I-A}(r))$ for r, and $1/2 \cdot \Delta w_{\text{right}}(d_{I-A}(l))$ for l, and the branch where $N_{I-A}[s]$ is deleted has a decrease in measure of $2w_{\text{sep}}(3)$ for s and s', $1/2 \cdot w_{\text{right}}(d_{I-A}(r))$ for r, $1/2 \cdot w_{\text{right}}(d_{I-A}(l))$ for l, and $(d_{I-A}(r) + d_{I-A}(l)) \cdot \delta_{\text{deg-dec}}$ for the vertices in $N_{I-A}(N_{I-A}[s])$. Thus, we get the following constraints for the balanced case,

$$\begin{aligned}&2^{-w_{\text{sep}}(3)-\Delta w_{\text{sep}}(3)-1/2\cdot(\Delta w_{\text{right}}(d_r)+\Delta w_{\text{right}}(d_l))}\\ &+2^{-2w_{\text{sep}}(3)-1/2\cdot(w_{\text{right}}(d_r)+w_{\text{right}}(d_l))-(d_r+d_l)\cdot\delta_{\text{deg-dec}}}\\ &\le 1, \qquad\qquad 2 \le d_l, d_r \le 3.\end{aligned} \tag{49}$$

In the imbalanced case, we obtain the following set of constraints, which are no more constraining than the previous set,

$$\begin{aligned}&2^{-w_{\text{sep}}(3)-\Delta w_{\text{sep}}(3)-\Delta w_{\text{right}}(d_r)}\\ &+2^{-2w_{\text{sep}}(3)-w_{\text{right}}(d_r)-d_r\cdot\delta_{\text{deg-dec}}}\\ &\le 1, \qquad\qquad 2 \le d_r \le 3.\end{aligned}$$

Here, the number of vertices in $N_{I-A}(N_{I-A}[s]) \cap (S \cup R)$ is d_r, accounting for the vertex r' and the neighbors of r besides s.

Now, consider the case where s has two neighbors in L or in R. Since the graph is bipartite, these two neighbors are not adjacent. If $\mu_r(R) - \mu_r(L) \le B$, then the two created subinstances are balanced, which gives us the following set of constraints:

$$\begin{aligned}&2^{-w_{\text{sep}}(3)-1/2\cdot(\Delta w_{\text{right}}(d_1)+\Delta w_{\text{right}}(d_2)+\Delta w_{\text{right}}(d_3))}\\ &+2^{-w_{\text{sep}}(3)-1/2\cdot(w_{\text{right}}(d_1)+w_{\text{right}}(d_2)+w_{\text{right}}(d_3))-(d_1+d_2+d_3-3)\cdot\delta_{\text{deg-dec}}}\\ &\le 1, \qquad\qquad 2 \le d_1, d_2, d_3 \le 3.\end{aligned} \tag{50}$$

Otherwise, $\mu_r(R) - \mu_r(L) > B$. If s has two neighbors in R, then the worst case is the balanced one, which is covered by (50). Finally, if s has two neighbors in L, then its neighbor $r \in R$ has $d_{I-A}(r) = 2$

due to line 17. The imbalanced case is not covered by (50) and incurs the following constraint on the measure.

$$2 \cdot 2^{-w_{\text{sep}}(3)-w_{\text{right}}(2)-\Delta w_{\text{right}}(d)} \leq 1, \qquad 2 \leq d \leq 3. \tag{51}$$

Note that the deletion of s triggers that r is removed by line 6 in Algorithm 2, which decreases the degree of its other neighbor, which is in R due to line 19.

This gives us all the constraints on μ_3 for the analysis. To obtain values for the various weights, we set

$$w_{\text{elt}}(i) = w_{\text{elt}}(i+1), \qquad w_{\text{set}}(i) = w_{\text{set}}(i+1), \qquad i \geq 6, \tag{52}$$

and we minimize $w_{\text{set}}(6) + w_{\text{elt}}(6)$. Since the separation has not added any non-convex constraints, the resulting mathematical program is convex, as usual [30]. Solving this convex program gives the following values for the weights:

$$\begin{aligned} w_{\text{right}}(2) &= 0.15282 & w_{\text{elt}}(2) &= 0.15384 & w_{\text{set}}(2) &= 0.16408 \\ w_{\text{right}}(3) &= 0.22669 & w_{\text{elt}}(3) &= 0.22732 & w_{\text{set}}(3) &= 0.24592 \\ w_{\text{sep}}(2) &= 0.75630 & w_{\text{elt}}(4) &= 0.26684 & w_{\text{set}}(4) &= 0.29320 \\ w_{\text{sep}}(3) &= 0.78943 & w_{\text{elt}}(5) &= 0.29023 & w_{\text{set}}(5) &= 0.30224 \\ & & w_{\text{elt}}(6) &= 0.30019 & w_{\text{set}}(6) &= 0.30224 \end{aligned}$$

For subcubic instances, the depth of the search tree is bounded by a polynomial since each recursive call decreases the measure $(|S_2| + |S|) \cdot \frac{\mu_r(V(I))}{w_{\text{right}}(2)} + \left|\frac{\mu_r(R)-\mu_r(L)}{w_{\text{right}}(2)}\right|$ by at least 1, where S_2 is the set of vertices in S that have degree 2 in $I - A$. By Lemma 2.3, we conclude that subcubic instances are solved in time $O^*(2^{\mu_3})$. Now, using Lemma 4.2 with $\mu = \mu_4$, $\mu' = \mu_3$ and $\eta = |V(I) \setminus A|$, we conclude that the running time of the algorithm is upper bounded by $O^\star(2^{\mu_4})$. Thus, the algorithm solves #Dominating Set in time $O^\star\left(2^{(w_{\text{elt}}(6)+w_{\text{set}}(6))\cdot n}\right) = O^\star(1.5183^n)$, where n is the number of vertices.

Theorem 5.2. *Algorithm #SC solves #Set Cover in time $O^\star\left(2^{0.30019\cdot|\mathcal{U}|+0.30224\cdot|\mathcal{S}|}\right)$ and #Dominating Set in time $O(1.5183^n)$, using polynomial space.*

5.3 Comments on the #SC algorithm and its analysis

Let us make a few final comments on the algorithm and its analysis. First, while the algorithm of van Rooij always prefers to branch on an element when the graph contains both an element and a set of maximum degree, this is not always possible when the algorithm needs to branch on vertices in the separator. However, this disadvantage is overwhelmed by the gain due to the separator branching.

Second, a certain amount of care needs to be taken to make sure that the algorithm terminates. For example, if we had omitted "that has degree 3 in $I - A$" in line 17 of Algorithm 3, then lines 14 and 18 could have eternally dragged vertices alternately to the left and to the right.

Third, the initial Set Cover transformation is not absolutely necessary, since we may equivalently label the vertices as in Subsection 5.1. We can then assign a weight to each vertex which depends on its label and the number of neighbors with each label. However, our attempt resulted in an unmanageable number of constraints.[5] We tried several compromises to simplify the analysis (merge

[5]With 8Gb of memory, we were only able to handle around 1.25 million constraints using a 64-bit version of AMPL and the solver NPSOL, and such instances were solved in around 10 minutes. This was not even sufficient to analyze running times for degree-4 instances.

labels, simplify the degree-spectrum), and we found that, of these, the simplification corresponding exactly to the Set Cover translation performed best. Here, a graph vertex u corresponds to an element e, which encodes that the vertex needs to be dominated, and to a set X, which encodes that it can be in the dominating set. Thus, in the measure, the weight of u is the weight of e plus the weight of X. Therefore, the weight of u depends on the degree of e and the degree of X, but not on the label of u and the degree of u. It would be interesting to know whether this choice of weights compromises the optimality of the analysis.

Finally, we remark that it did not help the analysis to allow different weights for sets and elements in the subcubic analysis. To further improve the running time, it will not be sufficient to improve the subcubic case, since none of the degree-3 constraints turn out tight.

6 LIMITATIONS OF THE METHOD

In this section, we discuss when we can and cannot expect to achieve improved running times using the Separate, Measure and Conquer method. The most important feature an algorithm needs to have is that it eventually encounters instances where small balanced separators can be computed efficiently. This is the case for algorithms that branch on graphs of bounded degree in their final stages, but other means of obtaining graphs with relatively small separators are conceivable. This might include cases where the treewidth of the graph is bounded by a fraction of the number of vertices, where the instance has a small backdoor set [32] to bounded treewidth instances (as in [33] for incidence graphs of SAT formulae), and graphs with small treewidth modulators [24, 40].

The first limitation of our method occurs when a non-separator based algorithm is already so fast that branching on separators does not add an advantage. For example, the current fastest algorithm for Maximum Independent Set on subcubic graphs has worst-case running time $O(1.0836^n)$, where n is the number of vertices of the input graph [69]. Merely branching on all the vertices of the separator would take time $2^{n/6+o(n)}$ and $2^{1/6} \approx 1.1225 > 1.0836$. This full time will be required, e.g., if the computed separator happens to be an independent set, so that a decision for one vertex of the separator (put it in the independent set or not) does not have any direct implications for the other vertices in the separator. While this is a limitation of our method, it also motivates the study of balanced separators with additional properties. A small but relatively dense separator could be useful for Maximum Independent Set branching algorithms.

The second limitation is that our Separate, Measure and Conquer subroutines can often be replaced by treewidth-based subroutines [17], leading to smaller worst-case running times but exponential space usage. In fact, replacing the branching on graphs with small maximum degree by a treewidth-based dynamic programming algorithm, such as [63], is a well-known method [17] to improve the worst-case running time of an algorithm at the expense of exponential-space usage. For instance, the currently fastest exponential-space algorithm for #Dominating Set [53] uses a treewidth-based subroutine on bounded degree instances to improve the running time. It solves #Dominating Set in $O(1.5002^n)$ time and exponential space. Using only polynomial space is however recognized as a significant advantage, especially for algorithms that may run much faster on real-world instances than their worst-case running time bounds, and the field devotes attention to both categories [23]. An exponential space usage would quickly become a bottleneck in the execution of the algorithm.

7 CONCLUSION

We have presented a new method to analyze separator based branching algorithms within the Measure and Conquer framework. It uses a novel kind of measure that is able to take advantage of a global structure in the instance and amortize a sudden large gain, due to the instance decomposing into several independent subinstances, over a linear number of previous branchings.

The method provides opportunities and challenges in the design and analysis of other exponential-time branching algorithms: opportunities because we believe it is widely applicable, and challenges because it complicates the analysis and leads to more choices in the design of algorithms. As usual, finding a set of reductions leading to a small running time upper bound in a Measure and Conquer analysis remains an art. The method has also been used to obtain faster polynomial-space algorithms for counting independent sets and graph coloring [29].

Finally, our analysis might open the way for other measures relying on global structures of instances.

ACKNOWLEDGMENTS

The research was supported in part by the DIMACS 2006–2010 Special Focus on Discrete Random Systems, NSF grant DMS-0602942. Serge Gaspers is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DE120101761) and a Future Fellowship (FT140100048), and acknowledges support under the ARC's Discovery Projects funding scheme (DP150101134). The research was done in part at Dagstuhl Seminars 10441 (Exact Complexity of NP-hard problems, 2010) and 13331 (Exponential Algorithms: Algorithms and Complexity Beyond Polynomial Time, 2013), and was presented at the latter.

REFERENCES

[1] Dimitris Achlioptas and Gregory B. Sorkin. 2000. Optimal myopic algorithms for random 3-SAT. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*. IEEE Comput. Soc. Press, Los Alamitos, CA, 590–600.

[2] Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. 2005. Exponential Lower Bounds for the Running Time of DPLL Algorithms on Satisfiable Formulas. *Journal of Automated Reasoning* 35, 1-3 (2005), 51–72.

[3] Armin Biere and Carsten Sinz. 2006. Decomposing SAT Problems into Connected Components. *JSAT, Journal on Satisfiability, Boolean Modeling and Computation* 2, 1-4 (2006), 201–208.

[4] Daniel Binkele-Raible. 2010. *Amortized Analysis of Exponential Time and Parameterized Algorithms: Measure & Conquer and Reference Search Trees.* Ph.D. Dissertation. University of Trier.

[5] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. 2009. Set partitioning via inclusion-exclusion. *SIAM J. Comput.* 39, 2 (2009), 546–563.

[6] Hans L. Bodlaender. 1998. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science* 209, 1-2 (1998), 1–45.

[7] Hans L. Bodlaender and Ton Kloks. 1996. Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs. *Journal of Algorithms* 21, 2 (1996), 358–402.

[8] Manfred Cochefert. 2014. *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles sur les graphes et hypergraphes.* Ph.D. Dissertation. Université de Lorraine, Metz, France. In French.

[9] Jean-François Couturier. 2012. *Algorithmes exacts et exponentiels sur les graphes : énumération, comptage et optimisation.* Ph.D. Dissertation. Université de Lorraine, Metz, France. In French.

[10] Jean-François Couturier, Pinar Heggernes, Pim van 't Hof, and Dieter Kratsch. 2013. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. *Theoretical Computer Science* 487 (2013), 82–94. https://doi.org/10.1016/j.tcs.2013.03.026

[11] Jean-François Couturier, Romain Letourneur, and Mathieu Liedloff. 2015. On the number of minimal dominating sets on some graph classes. *Theoretical Computer Science* 562 (2015), 634–642. https://doi.org/10.1016/j.tcs.2014.11.006

[12] Rina Dechter and Robert Mateescu. 2007. AND/OR search spaces for graphical models. *Artificial Intelligence* 171, 2-3 (2007), 73–106.

[13] Rodney G. Downey and Michael R. Fellows. 2013. *Fundamentals of Parameterized Complexity.* Springer.

[14] Keith Edwards and Eric McDermid. 2015. A General Reduction Theorem with Applications to Pathwidth and the Complexity of MAX 2-CSP. *Algorithmica* 72, 4 (2015), 940–968. https://doi.org/10.1007/s00453-014-9883-7

[15] Keith J. Edwards. 2016. A faster polynomial-space algorithm for Max 2-CSP. *J. Comput. System Sci.* 82, 3 (2016), 536–550.

[16] David Eppstein. 2006. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Transactions on Algorithms* 2, 4 (2006), 492–509.

[17] Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. 2009. On Two Techniques of Combining Branching and Treewidth. *Algorithmica* 54, 2 (2009), 181–207.

[18] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. 2009. A measure & conquer approach for the analysis of exact algorithms. *J. ACM* 56, 5 (2009), 25:1–25:32.

[19] Fedor V. Fomin, Fabrizio Grandoni, Dieter Kratsch, Daniel Lokshtanov, and Saket Saurabh. 2013. Computing Optimal Steiner Trees in Polynomial Space. *Algorithmica* 65, 3 (2013), 584–604. https://doi.org/10.1007/s00453-012-9612-z

[20] Fedor V. Fomin, Fabrizio Grandoni, Daniel Lokshtanov, and Saket Saurabh. 2012. Sharp Separation and Applications to Exact and Parameterized Algorithms. *Algorithmica* 63, 3 (2012), 692–706. https://doi.org/10.1007/s00453-011-9555-9

[21] Fedor V. Fomin, Fabrizio Grandoni, Artem V. Pyatkin, and Alexey A. Stepanov. 2008. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms* 5, 1 (2008), 9:1–9:17. https://doi.org/10.1145/1435375.1435384

[22] Fedor V. Fomin and Kjartan Høie. 2006. Pathwidth of cubic graphs and exact algorithms. *Inform. Process. Lett.* 97, 5 (2006), 191–196.

[23] Fedor V. Fomin and Dieter Kratsch. 2010. *Exact Exponential Algorithms.* Springer.

[24] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. 2012. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*. IEEE Computer Society, 470–479.

[25] Eugene C. Freuder and Michael J. Quinn. 1985. Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI 1985)*. Morgan Kaufmann, 1076–1078.

[26] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)*. ACM, 296–303. https://doi.org/10.1145/2608628.2608664

[27] Serge Gaspers. 2010. *Exponential Time Algorithms - Structures, Measures, and Bounds.* VDM. 206 pages.

[28] Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Ioan Todinca. 2009. Exponential time algorithms for the minimum dominating set problem on some graph classes. *ACM Transactions on Algorithms* 6, 1 (2009), 9:1–9:21. https://doi.org/10.1145/1644015.1644024

[29] Serge Gaspers and Edward J. Lee. 2017. Faster Graph Coloring in Polynomial Space. In *Proceedings of the 23rd Annual International Computing and Combinatorics Conference (COCOON 2017) (LNCS)*, Vol. 10392. Springer.

[30] Serge Gaspers and Gregory B. Sorkin. 2012. A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between. *J. Comput. System Sci.* 78, 1 (2012), 305–335. https://doi.org/10.1016/j.jcss.2011.05.010

[31] Serge Gaspers and Gregory B. Sorkin. 2015. Separate, Measure and Conquer: Faster Polynomial-Space Algorithms for Max 2-CSP and Counting Dominating Sets. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), Part I (Lecture Notes in Computer Science)*, Vol. 9134. Springer, 567–579. https://doi.org/10.1007/978-3-662-47672-7_46

[32] Serge Gaspers and Stefan Szeider. 2012. Backdoors to Satisfaction. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday (Lecture Notes in Computer Science)*, Vol. 7370. Springer, 287–317. https://doi.org/10.1007/978-3-642-30891-8_15

[33] Serge Gaspers and Stefan Szeider. 2013. Strong Backdoors to Bounded Treewidth SAT. In *Proceedings of the 54rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*. IEEE Computer Society, 489–498.

[34] Petr A. Golovach, Pinar Heggernes, and Dieter Kratsch. 2016. Enumerating minimal connected dominating sets in graphs of bounded chordality. *Theoretical Computer Science* 630 (2016), 63–75. https://doi.org/10.1016/j.tcs.2016.03.026

[35] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2000. A comparison of structural CSP decomposition methods. *Artificial Intelligence* 124, 2 (2000), 243–282.

[36] Fabrizio Grandoni. 2004. *Exact algorithms for hard graph problems.* Ph.D. Dissertation. University of Rome Tor Vergata, Italy.

[37] Fabrizio Grandoni. 2006. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms* 4, 2 (2006), 209–214. https://doi.org/10.1016/j.jda.2005.03.002

[38] Yoichi Iwata. 2012. 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. Springer, 41–54.

[39] Richard M. Karp. 1982. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters* 1, 2 (1982), 49–51.

[40] Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. 2013. Linear Kernels and Single-Exponential Algorithms via Protrusion Decompositions. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP 2013) (Lecture Notes in Computer Science)*, Vol. 7965. Springer, 613–624.

[41] Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. 2005. Algorithms Based on the Treewidth of Sparse Graphs. In *Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005)*. Lecture Notes in Computer Science, Vol. 3787. Springer, Berlin, 385–396.

[42] A. V. Kostochka and L. S. Melnikov. 1993. On a Lower Bound for the Isoperimetric Number of Cubic Graphs. In *Probablilistic Methods in Discrete Mathematics: Proceedings of the Third Petrozavodsk Conference*. TVP/VSP, 251–265.

[43] Marcin Krzywkowski. 2013. Trees having many minimal dominating sets. *Inform. Process. Lett.* 113, 8 (2013), 276–279. https://doi.org/10.1016/j.ipl.2013.01.020

[44] Wei Li and Peter van Beek. 2004. Guiding Real-World SAT Solving with Dynamic Hypergraph Separator Decomposition. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*. IEEE Computer Society, 542–548.

[45] Mathieu Liedloff. 2007. *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : domination, variantes et généralisations*. Ph.D. Dissertation. University of Metz, France. in French.

[46] Mathieu Liedloff. 2008. Finding a dominating set on bipartite graphs. *Inform. Process. Lett.* 107 (2008), 154–157.

[47] Richard J. Lipton and Robert Endre Tarjan. 1977. Application of a Planar Separator Theorem. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*. IEEE Computer Society, 162–170.

[48] Daniel Lokshtanov and Jesper Nederlof. 2010. Saving space by algebraization. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*. ACM, 321–330. https://doi.org/10.1145/1806689.1806735

[49] Burkhard Monien and Robert Preis. 2001. Upper Bounds on the Bisection Width of 3- and 4-Regular Graphs. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001) (Lecture Notes in Computer Science)*, Vol. 2136. Springer, 524–536.

[50] Burkhard Monien and Robert Preis. 2006. Upper bounds on the bisection width of 3- and 4-regular graphs. *Journal of Discrete Algorithms* 4, 3 (2006), 475–498. https://doi.org/10.1016/j.jda.2005.12.009

[51] Jesper Nederlof. 2013. Fast Polynomial-Space Algorithms Using Inclusion-Exclusion. *Algorithmica* 65, 4 (2013), 868–884. https://doi.org/10.1007/s00453-012-9630-x

[52] Jesper Nederlof and Johan M. M. van Rooij. 2010. Inclusion/Exclusion Branching for Partial Dominating Set and Set Splitting. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation (IPEC 2010) (Lecture Notes in Computer Science)*, Vol. 6478. Springer, 204–215. https://doi.org/10.1007/978-3-642-17493-3_20

[53] Jesper Nederlof, Johan M. M. van Rooij, and Thomas C. van Dijk. 2014. Inclusion/Exclusion Meets Measure and Conquer. *Algorithmica* 69, 3 (2014), 685–740. https://doi.org/10.1007/s00453-013-9759-2

[54] Neil Robertson and Paul D. Seymour. 1986. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms* 7, 3 (1986), 309–322.

[55] John M. Robson. 1986. Algorithms for maximum independent sets. *Journal of Algorithms* 7, 3 (1986), 425–440.

[56] Alexander D. Scott and Gregory B. Sorkin. 2006. Solving Sparse Random Instances of Max Cut and Max 2-CSP in Linear Expected Time. *Combinatorics Probability and Computing* 15, 1-2 (2006), 281–315. https://doi.org/10.1017/S096354830500725X

[57] Alexander D. Scott and Gregory B. Sorkin. 2007. Linear-Programming Design and Analysis of Fast Algorithms for Max 2-CSP. *Discrete Optimization* 4, 3-4 (2007), 260–287. https://doi.org/10.1016/j.disopt.2007.08.001

[58] Alexander D. Scott and Gregory B. Sorkin. 2009. Polynomial constraint satisfaction problems, graph bisection, and the Ising partition function. *ACM Transactions on Algorithms* 5, 4 (2009), Art. 45, 27. https://doi.org/10.1145/1597036.1597049

[59] Alexey A. Stepanov. 2008. *Exact algorithms for hard listing, counting and decision problems*. Ph.D. Dissertation. University of Bergen, Norway.

[60] Johan M. M. van Rooij. 2010. Polynomial Space Algorithms for Counting Dominating Sets and the Domatic Number. In *Proceedings of the 7th International Conference on Algorithms and Complexity (CIAC 2010) (Lecture Notes in Computer Science)*, Vol. 6078. Springer, 73–84.

[61] Johan M. M. van Rooij. 2011. *Exact Exponential-Time Algorithms for Domination Problems in Graphs*. Ph.D. Dissertation. Universiteit Utrecht.

[62] Johan M. M. van Rooij and Hans L. Bodlaender. 2011. Exact algorithms for dominating set. *Discrete Applied Mathematics* 159, 17 (2011), 2147–2164.

[63] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. 2009. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009) (Lecture Notes in Computer Science)*, Vol. 5757. Springer, 566–577.

[64] Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk. 2009. Inclusion/Exclusion Meets Measure and Conquer. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009) (Lecture Notes in Computer Science)*, Vol. 5757. Springer, 554–565.

[65] Magnus Wahlström. 2004. Exact algorithms for finding minimum transversals in rank-3 hypergraphs. *Journal of Algorithms* 51, 2 (2004), 107–121.

[66] Magnus Wahlström. 2007. *Algorithms, measures and upper bounds for satisfiability and related problems*. Ph.D. Dissertation. Linköping University, Sweden.

[67] Ryan Williams. 2005. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348, 2-3 (2005), 357–365.

[68] Gerhard J. Woeginger. 2004. Space and Time Complexity of Exact Algorithms: Some Open Problems. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC 2004) (Lecture Notes in Computer Science)*, Vol. 3162. Springer, 281–290. https://doi.org/10.1007/978-3-540-28639-4_25

[69] Mingyu Xiao and Hiroshi Nagamochi. 2013. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science* 469 (2013), 92–104.

Received August 2015; revised March 2017; accepted June 2017