Minimum Stable Cut and Treewidth

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France
michail.lampis@lamsade.dauphine.fr

Abstract

A stable or locally-optimal cut of a graph is a cut whose weight cannot be increased by changing the side of a single vertex. Equivalently, a cut is stable if all vertices have the (weighted) majority of their neighbors on the other side. Finding a stable cut is a prototypical PLS-complete problem that has been studied in the context of local search and of algorithmic game theory.

In this paper we study Min Stable Cut, the problem of finding a stable cut of minimum weight, which is closely related to the Price of Anarchy of the Max Cut game. Since this problem is NP-hard, we study its complexity on graphs of low treewidth, low degree, or both. We begin by showing that the problem remains weakly NP-hard on severely restricted trees, so bounding treewidth alone cannot make it tractable. We match this hardness with a pseudo-polynomial DP algorithm solving the problem in time $(\Delta \cdot W)^{O(\mathrm{tw})} n^{O(1)}$, where tw is the treewidth, Δ the maximum degree, and W the maximum weight. On the other hand, bounding Δ is also not enough, as the problem is NP-hard for unweighted graphs of bounded degree. We therefore parameterize Min Stable Cut by both tw and Δ and obtain an FPT algorithm running in time $2^{O(\Delta \mathrm{tw})}(n + \log W)^{O(1)}$. Our main result for the weighted problem is to provide a reduction showing that both aforementioned algorithms are essentially optimal, even if we replace treewidth by pathwidth: if there exists an algorithm running in $(nW)^{o(\mathrm{pw})}$ or $2^{o(\Delta \mathrm{pw})}(n + \log W)^{O(1)}$, then the ETH is false. Complementing this, we show that we can, however, obtain an FPT *approximation scheme* parameterized by treewidth, if we consider almost-stable solutions, that is, solutions where no single vertex can unilaterally increase the weight of its incident cut edges by more than a factor of $(1 + \varepsilon)$.

Motivated by these mostly negative results, we consider Unweighted Min Stable Cut. Here our results already imply a much faster exact algorithm running in time $\Delta^{O(\mathrm{tw})} n^{O(1)}$. We show that this is also probably essentially optimal: an algorithm running in $n^{o(\mathrm{pw})}$ would contradict the ETH.

2012 ACM Subject Classification Mathematics of computing→Graph algorithms; Theory of Computation → Design and Analysis of Algorithms → Parameterized Complexity and Exact Algorithms

Keywords and phrases Treewidth, Local Max-Cut, Nash Stability

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding The author is supported by ANR JCJC project "ASSK" (ANR-18-CE40-0025-01)

1 Introduction

In this paper we study problems related to *stable cuts* in graphs. A *stable* cut of an edge-weighted graph $G = (V, E)$ is a partition of V into two sets V_0, V_1 that satisfies the following property: for each $i \in \{0, 1\}$ and $v \in V_i$, the total weight of edges incident on v whose other endpoint is in V_{1-i} is at least half the total weight of all edges incident on v. In other words, a cut is stable if all vertices have the (weighted) majority of their incident edges cut.

The notion of stable cuts has been very widely studied from two different points of view. First, in the context of local search, a stable cut is a locally optimal cut: switching the side of any single vertex cannot increase the total weight of the cut. Hence, stable cuts have been studied with the aim to further our understanding of the basic local search heuristic for Max Cut. Second, in the context of algorithmic game theory a Max Cut game has often been considered, where each vertex is an agent whose utility is the total weight of edges connecting it to the other side. In this game, a stable cut corresponds exactly to the notion of a Nash equilibrium, that is, a state where no agent has an incentive to change her choice.

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The complexity of producing a Nash stable or locally optimal cut of a given edge-weighted graph has been heavily studied under the name Local Max Cut. The problem is known to be PLS-complete, under various restrictions (we give detailed references below).

In this paper we focus on a different but closely related optimization problem: given an edge-weighted graph we would like to produce a stable cut *of minumum total weight.* We call this problem Min Stable Cut. In addition to being a fairly natural problem on its own, we believe that Min Stable Cut is interesting from the perspective of both local search and algorithmic game theory. In the context of local search, Min Stable Cut is the problem of bounding the performance of the local search heuristic on a particular instance. It is folklore (and easy to see) that in general there exist graphs where the smallest stable cut has size half the maximum cut (e.g. consider a C_4) and this is tight since any stable cut must cut at least half the total edge weight. However, for most graphs this bound is far from tight. Min Stable Cut therefore essentially asks to estimate the ratio between the largest and smallest stable cut for a given specific instance. Similarly, in the context of algorithmic game theory, solving Min Stable Cut is essentially equivalent to calculating the Price of Anarchy of the Max Cut game on the given instance, that is, the ratio between the smallest stable cut and the maximum cut. Since we will mostly focus on cases where Max Cut is tractable, Min Stable Cut can, therefore, be seen as the problem of computing either the approximation ratio of local search or the Price of Anarchy of the Max Cut game on a given graph.

Our results It appears that little is currently known about the complexity of Min Stable Cut. However, since finding a (not necessarily minimum) stable cut is PLS-complete, finding the minimum such cut would be expected to be hard. Our focus is therefore to study the parameterized complexity of Min Stable Cut using structural parameters such as treewidth and the maximum degree of the input graph[1]. Our results are the following.

- First, we show that bounding only one of the two mentioned parameters is not sufficient to render the problem tractable. This is not suprising for the maximum degree Δ, where a reduction from Max Cut allows us to show the problem is NP-hard for $\Delta \leq 6$ even in the unweighted case (Theorem 4). It is, however, somewhat more disappointing that bounded treewidth also does not help, as the problem remains weakly NP-hard on trees of diameter 4 (Theorem 1) and bipartite graphs of vertex cover 2 (Theorem 3).
- These hardness results point to two directions for obtaining algorithms for Min Stable Cut: first, since the problem is "only" weakly NP-hard for bounded treewidth one could hope to obtain a pseudo-polynomial time algorithm in this case. We show that this is indeed possible and the problem is solvable in time $(\Delta \cdot W)^{O(\text{tw})} n^{O(1)}$, where W is the maximum edge weight (Theorem 5). Second, one may hope to obtain an FPT algorithm when both tw and Δ are parameters. We show that this is also possible and obtain an algorithm with complexity $2^{O(\Delta \text{tw})}(n + \log W)^{O(1)}$ (Theorem 6).
- These two algorithms lead to two further questions. First, can the $(\Delta \cdot W)^{O(\text{tw})} n^{O(1)}$ algorithm be improved to an FPT dependence on tw, that is, to running time $f(\text{tw})(nW)^{O(1)}$? And second, can the $2^{\Delta \text{tw}}$ parameter dependence of the FPT algorithm be improved, for example to $2^{O(\Delta + \text{tw})}$ or even $\Delta^{O(\text{tw})}$? We show that the answer to both questions is negative, even if we replace treewidth with pathwidth: under the ETH there is no algorithm running in $(nW)^{o(\text{pw})}$ or $2^{o(\Delta \text{tw})}(n + \log W)^{O(1)}$ (Theorem 8).

[1] We assume familiarity with the basics of parameterized complexity as given in standard textbooks [22].

- Complementing the above, we show that the problem does become FPT by treewidth alone if we allow the notion of approximation to be used in the concept of stability: there exists an algorithm which, for any $\varepsilon > 0$, runs in time $(\mathrm{tw}/\varepsilon)^{O(\mathrm{tw})}(n + \log W)^{O(1)}$ and produces a cut with the following properties: all vertices are $(1+\epsilon)$-stable, that is, no vertex can unilaterally increase its incident cut weight by more than a factor of $(1+\varepsilon)$; the cut has weight at most equal to that of the minimum stable cut.
- Finally, motivated by the above mostly negative results, we also consider Unweighted Min Stable Cut, the restriction of the problem where all edge weights are uniform. Our previous results give a much faster algorithm with parameter dependence $\Delta^{O(\mathrm{tw})}$, rather than $2^{\Delta \mathrm{tw}}$ (Corollary 12). However, this poses the natural question if in this case the problem finally becomes FPT by treewidth alone. Our main result in this part is to answer this question in the negative and show that, under the ETH, Unweighted Min Stable Cut cannot be solved in time $n^{o(\mathrm{pw})}$ (Theorem 13).

Taken together, our results paint a detailed picture of the complexity of Min Stable Cut parameterized by tw and Δ. All our exact algorithms (Theorems 5, 6) are obtained using standard dynamic programming on tree decompositions, the only minor complication being that for Theorem 6 we edit the decomposition to make sure that for each vertex some bag contains all of its neighborhood (this helps us verify that a cut is stable). The main technical challenge is in proving our complexity lower bounds. It is therefore perhaps somewhat surprising that the lower bounds turn out to be essentially tight, as this indicates that for Min Stable Cut and Unweighted Min Stable Cut, the straightforward DP algorithms are essentially optimal, if one wants to solve the problem exactly.

For the approximation algorithm, we rely on two rounding techniques: one is a rounding step similar to the one that gives an FPTAS for Knapsack by truncating weights so that the maximum weight is polynomially bounded. However, Min Stable Cut is more complicated than Knapsack, as an edge which is light for one of its endpoints may be heavy for the other. We therefore define a more general version of the problem, allowing us to decouple the contribution each edge makes to the stability of each endpoint. This helps us bound the largest stability-weight by a polynomial, but is still not sufficient to obtain an FPT algorithm, as the lower bound of Theorem 8 applies to polynomially bounded weights. We then go on to apply a technique introduced in [48] (see also [2, 10, 45, 46]) which allows us to obtain FPT approximation algorithms for problems which are W-hard by treewidth by applying a different notion of rounding to the dynamic program. This allows us to produce a solution that is simultaneously of optimal weight (compared to the best stable solution) and almost-stable, using essentially the same algorithm as in Theorem 5. However, it is worth noting that in general there is no obvious way to transform almost-stable solutions to stable solutions [12, 18], so our algorithm is not immediately sufficient to obtain an FPT approximation for Min Stable Cut if we insist on obtaining a cut which is exactly stable.

Related work From the point of view of local search algorithms, there is an extensive literature on the Local Max Cut problem, which asks us to find a stable cut (of any size). The problem has long been known to be PLS-complete [44, 54]. It remains PLS-complete for graphs of maximum degree 5 [28], but becomes polynomial-time solvable for graphs of maximum degree 3 [50, 53]. The problem remains PLS-complete if weights are assigned to vertices, instead of edges, and the weight of an edge is defined simply as the product of the weights of its endpoints [32]. Even though the problem is PLS-complete, it has long been observed that local search quickly finds a stable solution in most practical instances. One theoretical explanation for this phenomenon was given in a recent line of work which showed that Local Max Cut has quasi-polynomial time smoothed complexity [3, 13, 19, 30]. Local Max Cut is of course polynomial time solvable if all weights are polynomially bounded in n, as local improvements always increase the size of the cut.

In algorithmic game theory much work has been done on the complexity of computing Nash equilibria for the cut game and the closely related *party affiliation game*, in which players, represented by vertices, have to pick one of two parties and edge weights indicate how much two players gain if they are in the same party [6, 7, 20, 31, 37]. Note that for general graphical games finding an equilibrium is PPAD-hard on trees of constant pathwidth [26]. Because computing a stable solution is generally intractable, approximate equilibria have also been considered [12, 18]. Note that the notion of approximate equilibrium corresponds exactly to the approximation guarantee given by Theorem 11, but unlike the cited works, Theorem 11 produces a solution that is both approximately stable and as good as the optimal.

The problem we consider in this paper is more closely related to the problem of computing the *worst* (or best) Nash equilibrium, which in turn is closely linked to the notion of Price of Anarchy. For most problems in algorithmic game theory this type of question is usually NP-hard [14, 21, 27, 33, 36, 39, 55] and hard to approximate [5, 17, 23, 41, 51]. Even though these results show that finding a Nash equilibrium that maximizes an objective function is NP-hard under various restrictions (e.g. graphical games of bounded degree), to the best of our knowledge the complexity of finding the worst equilibrium of the Max Cut game (which corresponds to the Min Stable Cut problem of this paper) has not been considered.

Finally, another topic that has recently attracted attention in the literature is that of MinMax and MaxMin versions of standard optimization problems, where we search the worst solution which cannot be improved using a simple local search heuristic. The motivation behind this line of research is to provide bounds and a refined analysis of such basic heuristics. Problems that have been considered under this lens are Max Min Dominating Set [8, 25], Max Min Vertex Cover [16, 56],Max Min Separator [40], Max Min Cut [29], Min Max Knapsack [4, 34, 38], Max Min Edge Cover [47], Max Min FVS [24]. Some problems in this area also arise naturally in other forms and have been extensively studied, such as Min Max Matching (also known as Edge Dominating Set [43]) and Grundy Coloring, which can be seen as a Max Min version of Coloring [1, 9].

2 Definitions – Preliminaries

We generally use standard graph-theoretic notation and consider edge-weighted graphs, that is, graphs $G = (V, E)$ supplied with a weight function $w : E \to \mathbb{N}$. The weighted degree of a vertex $v \in V$ is $d_w(v) = \sum_{uv \in E} w(uv)$. A cut of a graph is a partition of V into V_0, V_1. A cut is *stable* for vertex $v \in V_i$ if $\sum_{vu \in E \wedge u \in V_{1-i}} w(vu) \geq \frac{d_w(v)}{2}$, that is, if the total weight of edges incident on v crossing the cut is at least half the weighted degree of v. In the Min Stable Cut problem we are given an edge-weighted graph and are looking for a cut that is stable for all vertices that minimizes the sum of weights of cut edges (that is, edges with endpoints on both sides of the cut). In Unweighted Min Stable Cut we restrict the problem so that the w function returns 1 for all edges. When describing stable cuts we will sometimes say that we "assign" value 0 (or 1) to a vertex; by this we mean that we place this vertex in V_0 (or V_1 respectively).

For the definitions of treewidth, pathwidth, and the related (nice) decompositions we refer to [22]. We will use as a complexity assumption the Exponential Time Hypothesis (ETH) [42] which states that there exists a constant $c > 1$ such that 3-SAT with n variables and m clauses cannot be solved in time c^{n+m}. In fact, we will use the slightly weaker and simpler form of the ETH which states that 3-SAT cannot be solved in time $2^{o(n+m)}$.

3 Weighted Min Stable Cut

In this section we present our results on exact algorithms for (weighted) MIN STABLE CUT. We begin with some basic NP-hardness reductions in Section 3.1, which establish that the problem remains (weakly) NP-hard when either the treewidth or the maximum degree are bounded. These set the stage for two algorithms, given in Section 3.2, solving the problem in pseudo-polynomial time for constant treewidth; and in FPT time parameterized by $\mathrm{tw} + \Delta$. In Section 3.3 we present a more fine-grained hardness argument, based on the ETH, which shows that the dependence on tw and Δ of our two algorithms is essentially optimal.

3.1 Basic Hardness Proofs

▶ **Theorem 1.** *MIN STABLE CUT is weakly NP-hard on trees of diameter* 4.

Proof. We describe a reduction from PARTITION. Recall that in this problem we are given n positive integers $x_1, \dots, x_n$ such that $\sum_{i=1}^{n} x_i = 2B$ and are asked if there exists $S \subseteq [n]$ such that $\sum_{i \in S} x_i = B$. We construct a star with n leaves and subdivide every edge once. For each $i \in [n]$ we select a distinct leaf of the tree and set the weight of both edges in the path from the center to this leaf to x_i. We claim that the graph has a stable cut of weight $3B$ if and only if there is a partition of $x_1, \dots, x_n$ into two sets with the same sum.

For the first direction, suppose $S \subseteq [n]$ is such that $\sum_{i \in S} x_i = B$. For each $i \in S$ we select a degree two vertex of the tree whose incident edges have weight x_i and assign it value 1. We assign all other degree two vertices value 0 and assign to all leaves the opposite of the value of their neighbor. We give the center value 0. This partition is stable as the center has edge weight exactly B towards each side, and all degree two vertices have a leaf attached that is placed on the other side and contributes half their total incident weight. The total weight cut is $2B$ from edges incident on leaves, plus B from half the weight incident on the center.

For the converse direction, observe that in any stable solution all edges incident on leaves are cut, contributing a weight of $2B$. As a result, in a stable cut of size $3B$, the weight of cut edges incident on the center is at most B. However, this weight is also at least B, since the edge weight incident on the center is $2B$. We conclude that the neighborhood of the center must be perfectly balanced. From this we can infer a solution to the PARTITION instance. ◀

▶ Remark 2. Theorem 1 is tight, because MIN STABLE CUT is trivial on trees of diameter at most 3.

▶ **Theorem 3.** *MIN STABLE CUT is weakly NP-hard on bipartite graphs with vertex cover* 2.

▶ **Theorem 4.** *UNWEIGHTED MIN STABLE CUT is strongly NP-hard and APX-hard on bipartite graphs of maximum degree* 6.

3.2 Algorithms

▶ **Theorem 5.** *There is an algorithm which, given an instance of MIN STABLE CUT with n vertices, maximum weight W, and a tree decomposition of width tw, finds an optimal solution in time $(\Delta \cdot W)^{O(\mathrm{tw})} n^{O(1)}$.*

▶ **Theorem 6.** *There is an algorithm which, given an instance of MIN STABLE CUT with n vertices, maximum weight W, maximum degree Δ and a tree decomposition of width tw, finds an optimal solution in time $2^{O(\Delta \mathrm{tw})}(n + \log W)^{O(1)}$.*

Proof. We describe an algorithm which works in a way similar to the standard algorithm for MAX CUT parameterized by treewidth, except that we work in a tree decomposition that is essentially a decomposition of the square of G. More precisely, before we begin, we do the following: for each $v \in V$ we add to every bag of the decomposition that contains v all the vertices of $N(v)$. It is not hard to see that we now have a decomposition of width at most $(\Delta+1)(\mathrm{tw}+1)$ and also that the new decomposition is still a valid tree decomposition. Crucially, we now also have the following property: for each $v \in V$ there exists at least one bag of the decomposition that contains all of $N[v]$.

The algorithm now performs dynamic programming by storing for each bag the value of the best solution for each partition of B_t. As a result, the size of the DP table is $2^{O(\Delta \mathrm{tw})}$. The only difference with the standard MAX CUT algorithm (beyond the fact that we are looking for a cut of minimum weight) is that when we consider a bag that contains all of $N[v]$, for some $v \in V$, we discard all partitions which are unstable for v. Since the bag contains all of $N[v]$, this can be checked in time polynomial in n and $\log W$ (assuming weights are given in binary). ◀

3.3 Tight ETH-based Hardness

We first give a reduction from 3-SET SPLITTING to MIN STABLE CUT whose main properties are laid out in Lemma 7. This reduction gives the lower bound of Theorem 8.

▶ **Lemma 7.** *There is a polynomial-time algorithm which, given a 3-SET SPLITTING instance $H = (V, E)$ with n elements, produces a MIN STABLE CUT instance G with the following properties: (i) G is a Yes instance if and only if H is a Yes instance; (ii) if Δ is the maximum degree of G and pw its pathwidth, then $\Delta = O(\log n)$ and $\mathrm{pw} = O(n/\log n)$; (iii) the maximum weight of G is $W = O(2^{\Delta})$.*

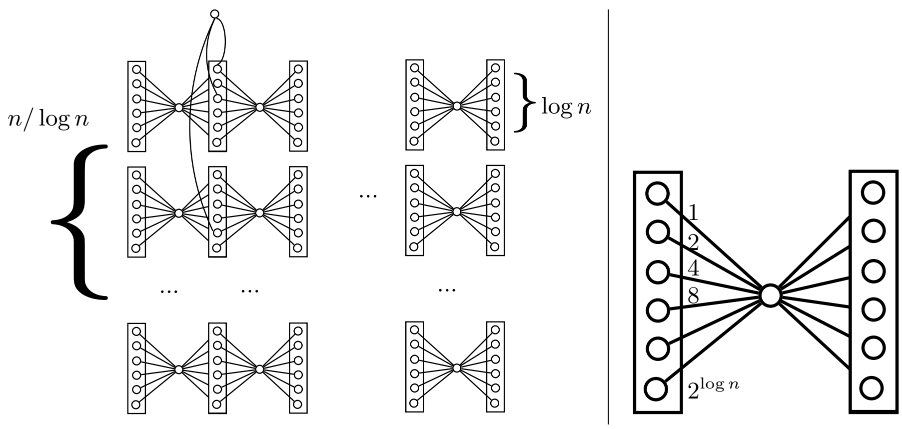

Figure 1 Sketch of the construction of Lemma 7. On the left, the general architecture: m columns, each with n vertices, partitioned into groups of size $\log n$. On each column we add a checker vertex (on top). Between the same groups of consecutive columns we add propagator vertices. On the right, more details about the exponentially increasing weights of edges incident on propagators.

Proof. Let $H = (V, E)$ be the given 3-SET SPLITTING instance, $V = \{v_0, \ldots, v_{n-1}\}$ and suppose that E contains e_2 sets of size 2 and e_3 sets of size 3, where $|E| = e_2 + e_3$ will be denoted by m. Assume without loss of generality that n is a power of 2 (otherwise add some dummy elements to V). Let $\delta = \log n$. We construct a graph by first making m copies of V, call them $V_j, j \in [m]$ and label their vertices as $V_j = \{v_{(i,j)} \mid i \in \{0, \ldots, n-1\}\}$. Intuitively, the vertices $\{v_{(i,j)} \mid j \in [m]\}$ are all meant to represent the element v_i of H. We now add to the graph the following:

1. Checkers: Suppose that the j-th set of E contains elements $v_{i_1}, v_{i_2}, v_{i_3}$. Then we construct a vertex c_j and connect it to $v_{(i_1,j)}, v_{(i_2,j)}, v_{(i_3,j)}$ with edges of weight 1. If the j-th set has size two, we do the same (ignoring v_{i_3}).
2. Propagators: For each $j \in [m-1]$ we construct $\rho = \lceil n/\delta \rceil$ vertices labeled $p_{(i,j)}, i \in \{0, \ldots, \rho-1\}$. Each $p_{(i,j)}$ is connected to (at most) δ vertices of V_j and δ vertices of V_{j+1} with edges of exponentially increasing weight. Specifically, for $i \in \{0, \ldots, \rho-1\}, \ell \in \{0, \ldots, \delta-1\}$, we connect $p_{(i,j)}$ to $v_{(i\delta+\ell,j)}$ and to $v_{(i\delta+\ell,j+1)}$ (if they exist) with an edge of weight 2^ℓ.
3. Stabilizers: For each $j \in [m], i \in \{0, \ldots, n-1\}$ we attach to $v_{(i,j)}$ a leaf. The edge connecting this leaf to $v_{(i,j)}$ has weight $3 \cdot 2^{(i \bmod \delta)}$.

This completes the construction of the graph. Let L be the total weight of edges incident on leaves and P be the total weight of edges incident on Propagator vertices $p_{(i,j)}$. We set $B = L + \frac{P}{2} + e_2 + 2e_3$ and claim that the new instance has a stable cut of weight B if and only if H can be split.

For the forward direction, suppose that H can be split by the partition of V into $L, R = V \setminus L$. We assign the following values for our new instance: for each $j \in [m]$ odd, we set $v_{(i,j)}$ to value 0 if and only if $v_i \in L$; for each $j \in [m]$ even, we set $v_{(i,j)}$ to value 0 if and only if $v_i \in R$. In other words, we use the same partition for all copies of V, but flip the roles of $0, 1$ between consecutive copies. We place leaves on the opposite side from their neighbors and greedily assign values to all other vertices of the graph to obtain a stable partition. Observe that all vertices $v_{(i,j)}$ are stable with the values we assigned, since the edge connecting each such vertex to a leaf has weight at least half its total incident weight.

In the partition we have we observe that (i) all edges incident on leaves are cut (total weight L) (ii) all Propagator vertices have balanced neighborhoods, so exactly half of their incident weight is cut (total weight $P/2$) (iii) since L, R splits all sets of E, each checker vertex will have exactly one neighbor on the same side (total weight $e_2 + 2e_3$). So the total weight of the cut is B.

For the converse direction, suppose we have a stable cut of size B in the constructed instance. Because of the stability condition, this solution must cut all edges incident on leaves (total weight L); at least half of the total weight of edges incident on Propagators (total weight $P/2$); and for each checker vertex all its incident edges except at most one (total weight at least $e_2 + 2e_3$). We conclude that, in order to achieve weight B, the cut must properly balance the neighborhood of all Propagators and make sure that each Checker vertex has one neighbor on its own side.

We now argue that because the neighborhood of each Propagator is balanced we have for all $i \in \{0, \ldots, n-1\}, j \in [m-1]$ that $v_{(i,j)}, v_{(i,j+1)}$ are on different sides of the partition. To see this, suppose for contradiction that for two such vertices this is not the case and to ease notation consider the vertices $v_{(i\delta+\ell,j)}, v_{(i\delta+\ell,j+1)}$, where $0 \le \ell \le \delta - 1$. Among all such pairs select one that maximizes ℓ. Both vertices are connected to the Propagator $p_{(i,j)}$ with edges of weight 2^ℓ. But now $p_{(i,j)}$ has strictly larger edge weight connecting it to the side of the partition that contains $v_{(i\delta+\ell,j)}$ and $v_{(i\delta+\ell,j+1)}$ than to the other side because (i) for neighbors of $p_{(i,j)}$ connected to it with edges of higher weight, the neighborhood of $p_{(i,j)}$ is balanced by the maximality of ℓ (ii) the total weight of all other edges is $2 \cdot (2^{\ell-1} + 2^{\ell-2} + \ldots + 1) < 2 \cdot 2^\ell$.

We thus have that for all i, j, $v_{(i,j)}, v_{(i,j+1)}$ must be on different sides, and therefore all V_j are partitioned in the same way (except perhaps with the role of 0 and 1 reversed). From this, we obtain a partition of V. To conclude this direction, we argue that this partition of V must split all sets. Indeed, if not, there will be a checker vertex such that all its neighbors are on the same side, which, as we argued, means that the cut must have weight strictly more than B.

Finally, let us show that the constructed instance has the claimed properties. The maximum degree is $\Delta = 2\delta = O(\log n)$ in the Propagators vertices (all other vertices have degree at most 4); the maximum weight is $O(2^\delta) = O(2^\Delta)$. Let us also consider the pathwidth of the constructed graph. Let G_j be the subgraph induced by V_j and its attached leaves, the Checker c_j, and all Propagators adjacent to V_j. We claim that we can build a path decomposition of G_j that contains all Propagators adjacent to V_j in all bags and has width $O(n/\log n)$. Indeed, if we place all the (at most $\lceil 2n/\delta \rceil$) Propagators and c_j in all bags, we can delete them from G_j, and all that is left is a union of isolated edges, which has pathwidth 1. Now, since the union of all G_j covers all vertices and edges, we can construct a path decomposition of the whole graph of width $O(n/\log n)$ by gluing together the decompositions of each G_j, that is, by connecting the last bag of the decomposition of G_j to the first bag of the decomposition of G_{j+1}. ◀

▶ **Theorem 8.** *If the ETH is true then (i) there is no algorithm solving* Min Stable Cut *in time $(nW)^{o(\mathrm{pw})}$ (ii) there is no algorithm solving* Min Stable Cut *in time $2^{o(\Delta \mathrm{pw})}(n + \log W)^{O(1)}$. These statements apply even if we restrict the input to instances where weights are written in unary and the maximum degree is $O(\log n)$.*

4 Approximately Stable Cuts

In this section we present an algorithm which runs in FPT time parameterized by treewidth and produces a solution that is $(1+\varepsilon)$-stable and has weight upper bounded by the weight of the optimal stable cut. Before we proceed, we will need to define a more general version of our problem. In Extended Min Stable Cut we are given as input: a graph $G = (V, E)$; a cut-weight function $w : E \to \mathbb{N}$; and a stability-weight function $s : E \times V \to \mathbb{N}$. For $v \in V$ we denote $d_s(v) = \sum_{vu \in E} s(vu, v)$, which we call the stability degree of v. If we are also given an error parameter $\rho > 1$, we will then be looking for a partition of V into V_0, V_1 which satisfies the following: (i) each vertex is ρ-stable, that is, for each $i \in \{0, 1\}$ and $v \in V_i$ we have $\sum_{vu \in E \wedge u \in V_{1-i}} s(vu, v) \geq \frac{d_s(v)}{2\rho}$ (ii) the total cut weight $\sum_{u \in V_0, v \in V_1, uv \in E} w(uv)$ is minimum. Observe that this extended version of the problem contains Min Stable Cut as a special case if $\rho = 1$ and for all $uv \in E$ we have $s(uv, v) = s(uv, u) = w(uv)$.

The generalization of Min Stable Cut is motivated by three considerations. First, the algorithm of Theorem 5 is inefficient because it has to store exact weight values to satisfy the stability constraints; however, it can efficiently store the total weight of the cut. We therefore decouple the contribution of an edge to the size of the cut (given by w) from a contribution of an edge to the stability of its endpoints (given by s). Second, our strategy will be to truncate the values of s so that the DP of the algorithm of Theorem 5 can be run more efficiently. To do this we will first simply divide all stability-weights by an appropriate value. However, a problem we run into if we do this is that the edge uv could simultaneously be one of the heavier edges incident on u and one of the lighter edges incident on v, so it is not clear how we can adjust its weight in a way that minimizes the distortion for both endpoints. As a result it is simpler if we allow edges to contribute different amounts to the stability of their endpoints. In this sense, $s(uv, u)$ is the amount that the edge uv contributes to the stability of u if the edge is cut. Observe that with the new definition, if we set a new stability-weight function for a specific vertex u as $s'(uv, v) = c \cdot s(uv, v)$ for all $v \in N(u)$, that is, if we multiply the stability-weight of all edges incident on u by a constant c and leave all other values unchanged, we obtain an equivalent instance, and this does not affect the stability of other vertices. Finally, the parameter ρ allows us to consider solutions where a vertex is stable if its cut incident edges are at least a $(\frac{1}{2\rho})$-fraction of its stability degree.

Armed with this intuition we can now explain our approach to obtaining our FPT approximation algorithm. Given an instance of the extended problem, we first adjust the s function so that its maximum value is bounded by a polynomial in n. We achieve this by dividing $s(uv, u)$ by a value that depends only on $d_s(u)$ and n. This allows us to guarantee that near-stable solutions are preserved. Then, given an instance where the maximum value of s is polynomially bounded, we apply the technique of [48], using the algorithm of Theorem 5 as a base, to obtain our approximation. We give these separate steps in the Lemmas below.

▶ **Lemma 9.** *There is an algorithm which, given a graph $G = (V, E)$ on n vertices and a stability-weight function $s : E \times V \to \mathbb{N}$ with maximum value S, runs in time polynomial in $n + \log S$ and produces a stability-weight function $s' : E \times V \to \mathbb{N}$ with the following properties: (i) the maximum value of s' is $O(n^2)$ (ii) for all partitions V into V_0, V_1, $i \in \{0, 1\}$, $v \in V_i$ we have*

$$\left(\frac{\sum_{vu \in E, u \in V_{1-i}} s(vu, v)}{d_s(v)}\right) / \left(\frac{\sum_{vu \in E, u \in V_{1-i}} s'(vu, v)}{d_{s'}(v)}\right) \in [1 - 1/n, 1 + 1/n]$$

Using Lemma 9 we can assume that all stability-weights are bounded by n^2. The most important part is that Lemma 9 guarantees us that almost-optimal solutions are preserved in both directions, as for any cut and for each vertex the ratio of stability weight going to the other side over the total stability-degree of the vertex does not change by more than a factor $(1 + \frac{1}{n})$. Let us now see the second ingredient of our algorithm.

▶ **Lemma 10.** *There is an algorithm which takes as input a graph $G = (V, E)$, a cut-weight function $w : E \to \mathbb{N}$ with maximum W, a stability-weight function $s : E \times V \to \mathbb{N}$ with maximum S, a tree decomposition of G of width tw, and an error parameter $\varepsilon > 0$ and returns a $(1+2\varepsilon)$-stable solution that has cut-weight at most equal to that of the minimum $(1+\epsilon)$-stable solution. If $S = O(n^2)$, then the algorithm runs in time $(\text{tw}/\varepsilon)^{O(\text{tw})}(n + \log W)^{O(1)}$.*

Proof. We use the methodology introduced in [48]. Before we proceed, let us explain that we are actually aiming for an algorithm with running time roughly $(\log n/\varepsilon)^{O(\text{tw})}$. This type of running time implies the time stated in the lemma using a standard Win/Win argument: if $\text{tw} \le \sqrt{\log n}$ then $(\log n)^{O(\text{tw})}$ is $n^{o(1)}$, so the $\log n^{O(\text{tw})}$ factor is absorbed in the $n^{O(1)}$ factor; while if $\log n \le \text{tw}^2$, then an algorithm running in $(\log n)^{\text{tw}}$ actually runs in $(\text{tw})^{O(\text{tw})}$.

To be more precise, if the given tree decomposition has height H, then we will formulate an algorithm with running time $(H \log S/\varepsilon)^{O(\text{tw})}(n + \log W)^{O(1)}$. This running time achieves parameter dependence $(\log n/\varepsilon)^{O(\text{tw})}$ if we use the fact that $S = O(n^2)$ and a theorem due to [15] which proves that any tree decomposition can be edited (in polynomial time) so that its height becomes $O(\log n)$, without increasing its width by more than a constant factor.

The basis of our algorithm will be the algorithm of Theorem 5, appropriately adjusted to the extended version of the problem. Let us first sketch the modifications to the algorithm of Theorem 5 that we would need to do to solve this more general problem, since the details are straightforward. First, we observe that in solution signatures we would now take into account stability-weights, and signatures would have values going up to S. Second, in Forget nodes, if we are happy with a $(1 + \varepsilon)$-solution, we would only discard solutions which violate this constraint. With these modifications, we can run this exact algorithm to return the minimum $(1+\varepsilon)$-stable solution in time $(2S)^{O(\mathrm{tw})}(n+\log W+\log(1/\varepsilon))^{O(1)}$.

The idea is to modify this algorithm so that the DP tables go from size $(2S)^{\mathrm{tw}}$ to roughly $(H\log S)^{\mathrm{tw}}$. To do this, we define a parameter $\delta=\frac{\varepsilon}{5H}$. We intend to replace every value x that would be stored in the signature of a solution in the DP table, with the next larger integer power of $(1+\delta)$, that is, to construct a DP table where x is replaced by $(1+\delta)^{\lceil\log_{(1+\delta)}x\rceil}$.

More precisely, the invariant we maintain is the following. Consider a node t of the decomposition at height h, where $h=0$ corresponds to leaves. We maintain a collection of solution signatures such that: (i) each signature contains a partition of B_t and for each $v\in B_t$ an integer that is upper-bounded by $\lceil\log_{(1+\delta)}d_s(v)\rceil$; (ii) Soundness: for each stored signature there exists a partition of $B_t^{\downarrow}$ which approximately corresponds to it. Specifically, the partition and the signature agree exactly on the assignment of B_t and the total cut-weight; the partition is $(1+2\varepsilon)$-stable for all vertices of $B_t^{\downarrow}\setminus B_t$; and for each $v\in B_t$, if the signature stores the value $x(v)$ for v, that is, it states that v has approximate stability-weight $(1+\delta)^{x(v)}$ towards its own side in $B_t^{\downarrow}\setminus B_t$, then in the actual partition the stability-weight of v to its own side of $B_t^{\downarrow}\setminus B_t$ is at most $(1+\delta)^h(1+\delta)^{x(v)}$. (iii) Completeness: conversely, for each partition of $B_t^{\downarrow}$ that is $(1+\varepsilon)$-stable for all vertices of $B_t^{\downarrow}\setminus B_t$ there exists a signature that approximately corresponds to it. Specifically, the partition and signature agree on the assignment of B_t and the total cut-weight; and for each $v\in B_t$, if the stability-weight of v towards its side of the partition of $B_t^{\downarrow}\setminus B_t$ is $y(v)$, and the signature stores the value $x(v)$, then $(1+\delta)^{x(v)}\le(1+\delta)^h y(v)$.

In more simple terms, the signatures in our DP table store values $x(v)$ so that we estimate that in the corresponding solution v has approximately $(1+\delta)^{x(v)}$ weight towards its own side in $B_t^{\downarrow}$, that is, we estimate that the DP of the exact algorithm would store approximately the value $(1+\delta)^{x(v)}$ for this solution. Of course, it is hard to maintain this relation exactly, so we are happy if for a node at height h the "true" value which we are approximating is at most a factor of $(1+\delta)^h$ off from our approximation.

Now, the crucial observation is that the approximate DP tables can be maintained because our invariant allows the error to increase with the height. For example, suppose that t is a Forget node at height h and let $u\in B_t$ be a neighbor of the vertex v we forget. The exact algorithm would construct the signature of a solution in t by looking at the signature of a solution in its child node, and then adding to the value stored for u the weight $s(vu,u)$ (if u,v are on the same side). Our algorithm will take an approximate signature from the child node, which may have a value at most $(1+\delta)^{h-1}$ the correct value, add to it $s(vu,u)$ and then, perhaps, round-up the value to an integer power of $(1+\delta)$. The new approximation will be at most $(1+\delta)^h$ larger than the value that the exact algorithm would have calculated. Similar argumentation holds for Join nodes. Furthermore, in Forget nodes we will only discard a solution if according to our approximation it is not $(1+2\varepsilon)$-stable. We may be over-estimating the stability-weight a vertex has to its own side of the cut by a factor of at most $(1+\delta)^h\le(1+\frac{\varepsilon}{5H})^H\le 1+\frac{\varepsilon}{2}$ so if for a signature our approximation says that the solution is not $(1+2\varepsilon)$-stable, the solution cannot be $(1+\varepsilon)$-stable, because $(1+\varepsilon)(1+\frac{\varepsilon}{2})<1+2\varepsilon$ (for sufficiently small ε).

Finally, to estimate the running time, the maximum value we have to store for each vertex in a bag is $\log_{(1+\delta)}S=\frac{\log S}{\log(1+\delta)}\le O(\frac{\log n}{\delta})=O(\frac{H\log n}{\varepsilon})$. Using the fact that $H=O(\log n)$ we get that the size of the DP table is $(\log n/\varepsilon)^{O(\mathrm{tw})}$. ◀

▶ **Theorem 11.** *There is an algorithm which, given an instance of* Min Stable Cut *$G=(V,E)$ with n vertices, maximum weight W, a tree decomposition of width* tw*, and a* *desired error $\varepsilon > 0$, runs in time $(\mathrm{tw}/\varepsilon)^{O(\mathrm{tw})}(n + \log W)^{O(1)}$ and returns a cut with the following properties: (i) for all $v \in V$, the total weight of edges incident on v crossing the cut is at least $(1-\varepsilon)\frac{d_w(v)}{2}$ (ii) the cut has total weight at most equal to the weight of the minimum stable cut.*

5 Unweighted Min Stable Cut

In this section we consider Unweighted Min Stable Cut. We first observe that applying Theorem 5 gives a parameter dependence of $\Delta^{O(\mathrm{tw})}$, since $W = 1$. We then show that this algorithm is essentially optimal, as the problem cannot be solved in $n^{o(\mathrm{pw})}$ under the ETH.

▶ **Corollary 12.** *There is an algorithm which, given an instance of Unweighted Min Stable Cut with n vertices, maximum degree Δ, and a tree decomposition of width tw, returns an optimal solution in time $\Delta^{O(\mathrm{tw})}n^{O(1)}$.*

We now first state our hardness result, then describe the construction of our reduction, and finally go through a series of lemmas that establish its correctness.

▶ **Theorem 13.** *If the ETH is true then no algorithm can solve Unweighted Min Stable Cut on graphs with n vertices in time $n^{o(\mathrm{pw})}$. Furthermore, Unweighted Min Stable Cut is W[1]-hard parameterized by pathwidth.*

To prove Theorem 13 we will describe a reduction from k-Multi-Colored Independent Set, a well-known W[1]-hard problem that cannot be solved in $n^{o(k)}$ time under the ETH [22]. Recall that in this problem we are given a graph $G = (V, E)$ with V partitioned into k color classes $V_1, \ldots, V_k$, each of size n, and we are asked to find an independent set of size k which selects one vertex from each V_i. In the remainder we use m to denote the number of edges of E and assume that vertices of V are labeled $v_{(i,j)}, i \in [k], j \in [n]$, where $V_i = \{v_{(i,j)} \mid j \in [n]\}$.

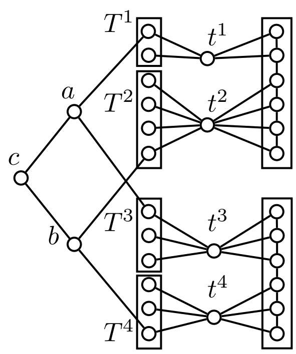

Figure 2 Checker gadget for Theorem 13. On the right two Selector gadgets. This Checker verifies that we have not taken an edge which has endpoints $(2, 3)$, hence t^1, t^3 are connected to the first 2 and 3 vertices of the Selectors.

Before we proceed, let us give some intuition. Our reduction will rely on a $k \times m$ grid-like construction, where each row represents the selection of a vertex in the corresponding color class of G and each column represents an edge of G. The main ingredients will be a Selector gadget, which will represent a choice of an index in $[n]$; a Propagator gadget which will make sure that the choice we make in each row stays consistent throughout; and a Checker gadget which will verify that we did not select the two endpoints of any edge. Each Selector gadget will contain a path on (roughly) n vertices such that any reasonable stable cut will have to cut exactly one edge of the path. The choice of where to cut this path will represent an index in $[n]$ encoding a vertex of G.

In our construction we will also make use of a simple but important gadget which we will call a "heavy" edge. Let $A = n^5$. When we say that we connect u, v with a heavy edge we will mean that we construct A new vertices and connect them to both u and v. The intuitive idea behind this gadget is that the large number of degree two vertices will force u and v to be on different sides of the partition (otherwise too many edges will be cut). We will also sometimes attach leaves on some vertices with the intention of making it easier for this vertex to achieve stability (as its attached leaves will always be on the other side of the partition).

Let us now describe our construction step-by-step.

1. Construct two "palette" vertices p_0, p_1 and a heavy edge connecting them. Note that all heavy edges we will add will be incident on at least one palette vertex.
2. For each $i \in [k], j \in [m]$ construct the following Selector gadget:
   a. Construct a path on $n+1$ vertices $P_{(i,j)}$ and label its vertices $P^1_{(i,j)}, \dots, P^{n+1}_{(i,j)}$.
   b. If j is odd, then add a heavy edge from $P^1_{(i,j)}$ to p_1 and a heavy edge from $P^{n+1}_{(i,j)}$ to p_0. If j is even, then add a heavy edge from $P^1_{(i,j)}$ to p_0 and a heavy edge from $P^{n+1}_{(i,j)}$ to p_1.
   c. Attach 5 leaves to each $P^{\ell}_{(i,j)}$ for $\ell \in \{2, \dots, n\}$. Attach $A+5$ leaves to $P^1_{(i,j)}$ and $P^{n+1}_{(i,j)}$.
3. For each $i \in [k], j \in [m-1]$ construct a new vertex connected to all vertices of the paths $P_{(i,j)}$ and $P_{(i,j+1)}$. This vertex is the Propagator gadget.
4. For each $j \in [m]$ consider the j-th edge of the original instance and suppose it connects $v_{(i_1,j_1)}$ to $v_{(i_2,j_2)}$. We construct the following Checker gadget (see Figure 2)
   a. We construct four vertices $t^1_j, t^2_j, t^3_j, t^4_j$. These are connected to existing vertices as follows: t^1_j is connected to $\{P^1_{(i_1,j)}, \dots, P^{j_1}_{(i_1,j)}\}$ (that is, the first j_1 vertices of the path $P_{(i_1,j)}$); t^2_j is connected to $\{P^{j_1+1}_{(i_1,j)}, \dots, P^{n+1}_{(i_1,j)}\}$ (that is, the remaining $n+1-j_1$ vertices of $P_{i_1,j}$); similarly, t^3_j is connected to $\{P^1_{(i_2,j)}, \dots, P^{j_2}_{(i_2,j)}\}$; and finally t^4_j is connected to $\{P^{j_2+1}_{(i_2,j)}, \dots, P^{n+1}_{(i_2,j)}\}$.
   b. We construct four independent sets $T^1_j, T^2_j, T^3_j, T^4_j$ with respective sizes $j_1, n+1-j_1, j_2, n+1-j_2$. We connect t^1_j to all vertices of T^1_j, t^2_j to T^2_j, t^3_j to T^3_j, and t^4_j to T^4_j. We attach two leaves to each vertex of $T^1_j \cup T^2_j \cup T^3_j \cup T^4_j$.
   c. We construct three vertices a_j, b_j, c_j. We connect c_j to both a_j and b_j. We connect a_j to an arbitrary vertex of T^1_j and an arbitrary vertex of T^3_j. We connect b_j to an arbitrary vertex of T^2_j and an arbitrary vertex of T^4_j.

Let L_1 be the number of leaves of the construction we described above and L_2 be the number of degree two vertices which are part of heavy edges. We set $B = L_1 + L_2 + km + k(m-1)(n+1) + m(2n+6)$.

▶ **Lemma 14.** *If G has a multi-colored independent set of size k, then the constructed instance has a stable cut of size at most B.*

▶ **Lemma 15.** *If the constructed instance has a stable cut of size at most B, then G has a multi-colored independent set of size k.*

▶ **Lemma 16.** *The constructed graph has pathwidth $O(k)$.*

6 Conclusions

Our results paint a clear picture of the complexity of Min Stable Cut with respect to tw and Δ. As directions for further work one could consider stronger notions of stability such as demanding that switching sets of k vertices cannot increase the cut, for constant k. We conjecture that, since the structure of this problem has the form $\exists\forall_k$, its complexity with respect to treewidth will turn out to be double-exponential in k [49]. Another direction is to consider *hedonic games* where vertices self-partition into an unbounded number of groups. The complexity of finding a stable solution in such games parameterized by tw + Δ has already been considered by Peters [52], whose algorithm runs in time exponential in Δ^5tw. Can we bridge the gap between this complexity and the $2^{O(\Delta \mathrm{tw})}$ complexity of Min Stable Cut?

References

1 Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring & friends, half-graphs, bicliques. In *STACS*, volume 154 of *LIPIcs*, pages 58:1–58:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

2 Eric Angel, Evripidis Bampis, Bruno Escoffier, and Michael Lampis. Parameterized power vertex cover. *Discret. Math. Theor. Comput. Sci.*, 20(2), 2018. URL: `http://dmtcs.episciences.org/4873`.

3 Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local max-cut in smoothed polynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 429–437. ACM, 2017. `doi:10.1145/3055399.3055402`.

4 Esther M. Arkin, Michael A. Bender, Joseph S. B. Mitchell, and Steven Skiena. The lazy bureaucrat scheduling problem. *Inf. Comput.*, 184(1):129–146, 2003.

5 Per Austrin, Mark Braverman, and Eden Chlamtac. Inapproximability of np-complete variants of nash equilibrium. *Theory Comput.*, 9:117–142, 2013. `doi:10.4086/toc.2013.v009a003`.

6 Baruch Awerbuch, Yossi Azar, Amir Epstein, Vahab S. Mirrokni, and Alexander Skopalik. Fast convergence to nearly optimal solutions in potential games. In Lance Fortnow, John Riedl, and Tuomas Sandholm, editors, *Proceedings 9th ACM Conference on Electronic Commerce (EC-2008), Chicago, IL, USA, June 8-12, 2008*, pages 264–273. ACM, 2008. `doi:10.1145/1386790.1386832`.

7 Maria-Florina Balcan, Avrim Blum, and Yishay Mansour. Improved equilibria via public service advertising. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 728–737. SIAM, 2009. URL: `http://dl.acm.org/citation.cfm?id=1496770.1496850`.

8 C. Bazgan, L. Brankovic, K. Casel, H. Fernau, K. Jansen, K.-M. Klein, M. Lampis, M. Liedloff, J. Monnot, and V. T. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018.

9 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ESA.2020.14`.

10 Rémy Belmonte, Michael Lampis, and Valia Mitsou. Parameterized (approximate) defective coloring. *SIAM J. Discret. Math.*, 34(2):1084–1106, 2020. `doi:10.1137/18M1223666`.

11 Piotr Berman and Marek Karpinski. On some tighter inapproximability results (extended abstract). In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 1999. `doi:10.1007/3-540-48523-6\_17`.

12 Anand Bhalgat, Tanmoy Chakraborty, and Sanjeev Khanna. Approximating pure nash equilibrium in cut, party affiliation, and satisfiability games. In David C. Parkes, Chrysanthos Dellarocas, and Moshe Tennenholtz, editors, *Proceedings 11th ACM Conference on Electronic Commerce (EC-2010), Cambridge, Massachusetts, USA, June 7-11, 2010*, pages 73–82. ACM, 2010. `doi:10.1145/1807342.1807353`.

13 Ali Bibak, Charles Carlson, and Karthekeyan Chandrasekaran. Improving the smoothed complexity of FLIP for max cut problems. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 897–916. SIAM, 2019. `doi:10.1137/1.9781611975482.55`.

14 Vittorio Bilò and Marios Mavronicolas. The complexity of computational problems about nash equilibria in symmetric win-lose games. *CoRR*, abs/1907.10468, 2019. URL: `http://arxiv.org/abs/1907.10468`, `arXiv:1907.10468`.

15 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998.

16 É. Bonnet, M. Lampis, and V. T. Paschos. Time-approximation trade-offs for inapproximable problems. *Journal of Computer and System Sciences*, 92:171 – 180, 2018.

17 Mark Braverman, Young Kun-Ko, and Omri Weinstein. Approximating the best nash equilibrium in $n^{o(\log n)}$-time breaks the exponential time hypothesis. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 970–982. SIAM, 2015. doi:10.1137/1.9781611973730.66.

18 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate pure nash equilibria in weighted congestion games: Existence, efficient computation, and structure. *ACM Trans. Economics and Comput.*, 3(1):2:1–2:32, 2015. doi:10.1145/2614687.

19 Xi Chen, Chenghao Guo, Emmanouil-Vasileios Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. Smoothed complexity of local max-cut and binary max-csp. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1052–1065. ACM, 2020. doi:10.1145/3357713.3384325.

20 George Christodoulou, Vahab S. Mirrokni, and Anastasios Sidiropoulos. Convergence and approximation in potential games. *Theor. Comput. Sci.*, 438:13–27, 2012. doi:10.1016/j.tcs.2012.02.033.

21 Vincent Conitzer and Tuomas Sandholm. New complexity results about nash equilibria. *Games Econ. Behav.*, 63(2):621–641, 2008. doi:10.1016/j.geb.2008.02.015.

22 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

23 Argyrios Deligkas, John Fearnley, and Rahul Savani. Inapproximability results for constrained approximate nash equilibria. *Inf. Comput.*, 262(Part):40–56, 2018. doi:10.1016/j.ic.2018.06.001.

24 Louis Dublois, Tesshu Hanaka, Mehdi Khosravian Ghadikolaei, Michael Lampis, and Nikolaos Melissinos. (in)approximability of maximum minimal FVS. In *ISAAC*, volume 181 of *LIPIcs*, pages 3:1–3:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

25 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. Upper dominating set: Tight algorithms for pathwidth and sub-exponential approximation. *CoRR*, abs/2101.07550, 2021. URL: https://arxiv.org/abs/2101.07550, arXiv:2101.07550.

26 Edith Elkind, Leslie Ann Goldberg, and Paul W. Goldberg. Nash equilibria in graphical games on trees revisited. In Joan Feigenbaum, John C.-I. Chuang, and David M. Pennock, editors, *Proceedings 7th ACM Conference on Electronic Commerce (EC-2006), Ann Arbor, Michigan, USA, June 11-15, 2006*, pages 100–109. ACM, 2006. doi:10.1145/1134707.1134719.

27 Edith Elkind, Leslie Ann Goldberg, and Paul W. Goldberg. Computing good nash equilibria in graphical games. In Jeffrey K. MacKie-Mason, David C. Parkes, and Paul Resnick, editors, *Proceedings 8th ACM Conference on Electronic Commerce (EC-2007), San Diego, California, USA, June 11-15, 2007*, pages 162–171. ACM, 2007. doi:10.1145/1250910.1250935.

28 Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2011. doi:10.1007/978-3-642-22006-7_15.

29 Hiroshi Eto, Tesshu Hanaka, Yasuaki Kobayashi, and Yusuke Kobayashi. Parameterized Algorithms for Maximum Cut with Connectivity Constraints. In *IPEC 2019*, pages 13:1–13:15, 2019.

30 Michael Etscheid and Heiko Röglin. Smoothed analysis of local search for the maximum-cut problem. *ACM Trans. Algorithms*, 13(2):25:1–25:12, 2017. doi:10.1145/3011870.

31 Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 604–612. ACM, 2004. doi:10.1145/1007352.1007445.

32 Dimitris Fotakis, Vardis Kandiros, Thanasis Lianeas, Nikos Mouzakis, Panagiotis Patsilinakos, and Stratis Skoulakis. Node-max-cut and the complexity of equilibrium in linear weighted congestion games. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 50:1–50:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.50.

33 Dimitris Fotakis, Spyros C. Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul G. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. *Theor. Comput. Sci.*, 410(36):3305–3326, 2009. doi:10.1016/j.tcs.2008.01.004.

34 F. Furini, I. Ljubić, and M. Sinnl. An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. *European Journal of Operational Research*, 262(2):438–448, 2017.

35 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

36 Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

37 Laurent Gourvès and Jérôme Monnot. On strong equilibria in the max cut game. In Stefano Leonardi, editor, *Internet and Network Economics, 5th International Workshop, WINE 2009, Rome, Italy, December 14-18, 2009. Proceedings*, volume 5929 of *Lecture Notes in Computer Science*, pages 608–615. Springer, 2009. doi:10.1007/978-3-642-10841-9_62.

38 Laurent Gourvès, Jérôme Monnot, and Aris Pagourtzis. The lazy bureaucrat problem with common arrivals and deadlines: Approximation and mechanism design. In *FCT*, volume 8070 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2013.

39 Gianluigi Greco and Francesco Scarcello. On the complexity of constrained nash equilibria in graphical games. *Theor. Comput. Sci.*, 410(38-40):3901–3924, 2009. doi:10.1016/j.tcs.2009.05.030.

40 Tesshu Hanaka, Hans L. Bodlaender, Tom C. van der Zanden, and Hirotaka Ono. On the maximum weight minimal separator. *Theoretical Computer Science*, 796:294 – 308, 2019.

41 Elad Hazan and Robert Krauthgamer. How hard is it to approximate the best nash equilibrium? *SIAM J. Comput.*, 40(1):79–91, 2011. doi:10.1137/090766991.

42 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

43 Ken Iwaide and Hiroshi Nagamochi. An improved algorithm for parameterized edge dominating set problem. *J. Graph Algorithms Appl.*, 20(1):23–58, 2016.

44 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

45 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

46 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discrete Applied Mathematics*, 2020. URL: http://www.sciencedirect.com/science/article/pii/S0166218X20301517, doi:10.1016/j.dam.2020.03.052.

47 Kaveh Khoshkhah, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Weighted upper edge cover: Complexity and approximability. *J. Graph Algorithms Appl.*, 24(2):65–88, 2020.

48 Michael Lampis. Parameterized approximation schemes using graph widths. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. doi:10.1007/978-3-662-43948-7_64.

49 Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.26.

50 Martin Loebl. Efficient maximal cubic graph cuts (extended abstract). In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, volume 510 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 1991. doi:10.1007/3-540-54233-7_147.

51 Lorenz Minder and Dan Vilenchik. Small clique detection and approximate nash equilibria. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 673–685. Springer, 2009. doi:10.1007/978-3-642-03685-9_50.

52 Dominik Peters. Graphical hedonic games of bounded treewidth. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 586–593. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12400.

53 Svatopluk Poljak. Integer linear programs and local search for max-cut. *SIAM J. Comput.*, 24(4):822–839, 1995. doi:10.1137/S0097539793245350.

54 Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991. doi:10.1137/0220004.

55 Grant Schoenebeck and Salil P. Vadhan. The computational complexity of nash equilibria in concisely represented games. *ACM Trans. Comput. Theory*, 4(2):4:1–4:50, 2012. doi:10.1145/2189778.2189779.

56 M. Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017.

A Omitted Material

A.1 Proof of Remark 2

Proof. A tree of diameter at most 3 must be either a star, in which case there is only one feasible solution (up to symmetry); or a double-star, that is a graph produced by taking two stars and connecting their centers. In the latter case, the optimal solution is always to place the two centers on the same side if this is feasible (as otherwise all edges are cut). ◀

A.2 Proof of Theorem 3

Proof. We present a reduction from PARTITION similar to that of Theorem 1. Given an instance with values $x_1, \ldots, x_n$ we construct a bipartite graph $K_{2,n}$. To ease presentation, we will call the part of $K_{2,n}$ that contains two vertices the "left" part, and the part that contains the remaining n vertices the "right" part. For each $i \in [n]$ we select a vertex of the right part and set the weight of both its incident edges to x_i. We claim that this graph has a stable cut of weight $2B$ if and only if the original instance is a Yes instance.

If there is a partition $S \subseteq [n]$ such that $\sum_{i \in S} x_i = B$, we select the corresponding vertices of the right part and assign to them 0; we assign 1 to the other vertices of the right part; we assign 0 to one vertex of the left part and 1 to the other. This partition is stable, as all vertices have completely balanced neighborhoods. Furthermore, the weight of the cut is $2B$.

For the other direction, observe that if both vertices of the left part of $K_{2,n}$ are on the same side of the partition, then all edges will be cut, giving weight $4B$. So a stable partition of weight $2B$ must place these two vertices on different sides. However, these vertices have the same neighbors (with the same edge weights), so if both are stable, their neighborhood must be properly balanced. From this we can infer a solution to the PARTITION instance. ◀

A.3 Proof of Theorem 4

Proof. We give a reduction from MAX CUT on graphs of maximum degree 3, which is known to be APX-hard [11]. Given an instance $G = (V, E)$ of MAX CUT we sub-divide each edge of E once, and we attach three leaves to each vertex of V. We claim that if the original instance has a cut of size at least k then the new instance has a stable cut of size at most $3|V| + 2|E| - k$.

For one direction, suppose we have a cut of G of size k which partitions V into V_0, V_1. We use the same partition of V for the new instance. For each leaf, we assign it a value opposite of that of its neighbor. For each degree two vertex which was produced when sub-dividing an edge of E we give it a value that is opposite to that of at least one of its neighbors. Observe that this cut is stable: all leaves are stable; all vertices produced in sub-divisions have degree two and at least one neighbor on the other side; and all vertices of V are adjacent to three leaves on the other side and at most three other vertices (since G is subcubic). The edges cut are: $3|V|$ edges incident on leaves; 2 edges for each edge of E whose endpoints are on the same side; 1 edge for each cut edge of E. This gives $3|V| + 2|E| - k$ edges cut overall.

For the other direction, suppose we have a stable cut of the new graph. We use the same cut in G and claim that it must cut at least k edges. Indeed, in the new graph any stable cut must cut all $3|V|$ edges incident on leaves, and at least one of the two edges incident on each degree two vertex. Furthermore, if $e = (u, v) \in E$ and u, v are on the same side of the cut, then both edges in the sub-divided edge e must be cut. We conclude that there must be at least k edges of G with endpoints on different sides of the cut. ◀

A.4 Proof of Theorem 5

Proof. We sketch some of the details, since our algorithm follows the standard dynamic programming method for treewidth. We assume that we are given a nice tree decomposition of width tw for the input graph $G = (V, E)$. For each node t of the decomposition, let $B_t \subseteq V$ be the bag associated with t and $B_t^{\downarrow} \subseteq V$ the set of all vertices of G which appear in bags in the sub-tree rooted in t (that is, the vertices which appear below t in the decomposition). The signature of a solution in node t is defined as a tuple of the following information: (i) a partition of B_t into two sets, which encodes the intersections of B_t with V_0, V_1 (ii) for each $v \in B_t$ an integer value in $\{0, \dots, d_w(v)\}$, which encodes for each $v \in B_t$ the total weight of its incident edges whose other endpoint is in $B_t^{\downarrow} \setminus B_t$ and on the same side of the cut as v. Our dynamic program stores in each node B_t, for each possible signature s, a value $c(s)$, which is the size of the best cut of $B_t^{\downarrow}$ that is consistent with the signature and is also stable for all vertices of $B_t^{\downarrow} \setminus B_t$. Observe that the total number of possible signatures is at most $2^{|B_t|}(\max d_w(v))^{|B_t|} \le O(2^{\mathrm{tw}}(\Delta \cdot W)^{\mathrm{tw}+1})$, because $d_w(v)$ is always upper-bounded by $\Delta \cdot W$. Therefore, what remains is to show that we can maintain the dynamic programming tables in time polynomial in their size. If we do this then it's not hard to see that examining the DP table of the root will allow us to find the optimal solution.

As mentioned, the basic idea of the algorithm is that for a node t of the decomposition and a signature s, we will maintain the value $c(s)$ if we have the following: (i) there exists a partition of $B_t^{\downarrow}$ into V_0, V_1 that is consistent with s in B_t, stable for all vertices of $B_t^{\downarrow} \setminus B_t$, such that the total weight of cut edges of $G[B_t^{\downarrow}]$ is $c(s)$; (ii) for any other partition of $B_t^{\downarrow}$ that is consistent with s in B_t and stable for all vertices of $B_t^{\downarrow} \setminus B_t$, the total weight of its cut edges of $G[B_t^{\downarrow}]$ is at least $c(s)$. To clarify what we mean that the partition is consistent with s in B_t, we recall that s specifies a partition of B_t with which the partition must agree; and furthermore s specifies for each $v \in B_t$ its total incident edge weight leading to the same side of the partition in $B_t^{\downarrow} \setminus B_t$ and the actual partition must also agree with these values.

Given the above framework, it's now not hard to complete the dynamic programming algorithm. For Leaf nodes, the table contains only the trivial signature, which has value 0, since the corresponding bag is empty. For Introduce nodes that add a new vertex v, we consider every signature of the child node and extend it by considering placing v into V_0 or V_1. Since all neighbors of v in $B_t^{\downarrow}$ are contained in B_t, placing v doesn't change the signature of other vertices and v has 0 weight to $B_t^{\downarrow} \setminus B_t$. For Forget nodes that remove a vertex v, we discard all signatures in which v has more then $d_w(v)/2$ of its incident weight going to its own side (since in such solution v will be unstable) and keep the remaining signatures, updating the weighted information of neighbors of v in B_t. Finally, for Join nodes, we only consider pairs of signatures which agree on the partition of B_t into V_0, V_1. For each such pair, we can compute the weighted degree of each v towards its side of the partition in $B_t^{\downarrow} \setminus B_t$, by adding the corresponding values in the two signatures. Observe that this doesn't double-count any edge, as edge induced by B_t are taken care of in Forget nodes. ◀

A.5 Proof of Theorem 8

Proof. We recall that the standard chain of reductions from 3-SAT to 3-Set Splitting which establishes that the latter problem is NP-hard produces an instance with size linear in the original formula [35, 42]. We compose these reductions with the reduction of Lemma 7. Suppose we started with a formula with n variables and m clauses (so as an intermediate step we constructed a 3-Set Splitting instance with $O(n+m)$ elements and sets). We therefore now have an instance with $N = poly(n+m)$ vertices (since the reduction runs in polynomial time), maximum degree $\Delta = O(\log(n+m))$ and pathwidth $\mathrm{pw} = O((n+m)/\log(n+m))$, and maximum weight $W = poly(n+m)$. Plugging these relations into the running times of hypothetical algorithms for Min Stable Cut we obtain algorithms for 3-SAT running in time $2^{o(n+m)}$ and contradicting the ETH. ◀

A.6 Proof of Lemma 9

Proof. For $v \in V$ let $S(v) = \max_{u\in N(v)} s(vu, v)$. We define s' as follows: $s'(vu, v) = \lfloor \frac{n^2 s(uv,v)}{S(v)} \rfloor$. It is clear that the maximum value of s' is n^2 and that calculations can be carried out in the promised time. So what remains is to prove that for any partition the fraction $\frac{\sum_{vu\in E, u\in V_{1-i}} s(vu,v)}{d_s(v)}$ stays essentially unchanged.

Observe that $\frac{n^2 s(uv,v)}{S(v)} \le s'(vu, v) \le \frac{n^2 s(uv,v)}{S(v)} + 1$. We therefore have

$$\frac{n^2 d_s(v)}{S(v)} \le d_{s'}(v) \le \frac{n^2 d_s(v)}{S(v)} + n$$

We also have:

$$\frac{n^2 \sum_{vu\in E, u\in V_{1-i}} s(vu,v)}{S(v)} \le \sum_{vu\in E, u\in V_{1-i}} s'(vu,v) \le \frac{n^2 \sum_{vu\in E, u\in V_{1-i}} s(vu,v)}{S(v)} + n$$

In both cases we have used the fact that the degree of v is at most n. Now with some calculation we get:

$$\frac{\sum_{vu\in E, u\in V_{1-i}} s(vu,v)}{d_s(v) + \frac{S(v)}{n}} \le \frac{\sum_{vu\in E, u\in V_{1-i}} s'(vu,v)}{d_{s'}(v)} \le \frac{\sum_{vu\in E, u\in V_{1-i}} s(vu,v) + \frac{S(v)}{n}}{d_s(v)}$$

We can now use the fact that $S(v) < d_s(v)$ and that $\frac{1}{1+\frac{1}{n}} > 1 - \frac{1}{n}$.

◀

A.7 Proof of Theorem 11

Proof. We simply put together the algorithms of Lemmas 9 and 10. Fix an $\varepsilon > 0$. Once we execute the algorithm of Lemma 9 the weight of all cuts is preserved (since we do not change w), and a stable cut remains at least $(1+\varepsilon/2)$-stable, if n is sufficiently large. We therefore execute the algorithm of Lemma 10 and this will output a $(1+\varepsilon)$-stable cut with value at least as small as the minimum stable cut. ◀

A.8 Proof of Lemma 14

Proof. Let $\sigma : [k] \to [n]$ be a function that encodes a multi-colored independent set of G, that is, the set $\{v_{(i,\sigma(i))} \mid i \in [k]\}$ is an independent set. We construct a partition of the new instance as follows: we assign 0 to p_0, 1 to p_1, and arbitrary values to the vertices of the heavy edge connecting p_0 to p_1; each other vertex that belongs to a heavy edge incident to p_0 (respectively p_1) is assigned 1 (respectively 0); each vertex connected via a heavy edge to p_0 (respectively p_1) is assigned 1 (respectively 0); for each Selector gadget $P_{(i,j)}$ we assign to the first $\sigma(i)$ vertices of the path (that is, the vertices $\{P^1_{i,j}, \ldots, P^{\sigma(i)}_{i,j}\}$) the same value as $P^1_{i,j}$ (that is, 0 if j is odd and 1 if j is even); we assign to the remaining vertices of $P_{(i,j)}$ the same value as $P^{n+1}_{(i,j)}$; we assign to every leaf the opposite value from that of its neighbor; we assign an arbitrary value to each Propagator vertex. We have now described a partition of all the vertices except of the non-leaf vertices belonging to Checker gadgets.

Before we describe the partition of the Checker gadgets let us establish some basic properties of the partition so far. First, all vertices for which we have given a value are stable, independent of the values we intend to assign to the non-leaf Checker gadget vertices. To see this we note that (i) all leaves have a value different from their neighbors (ii) all degree 2 vertices that belong to heavy edges have two neighbors with distinct values (iii) p_0 and p_1 have the majority of their neighbors on the other side of the partition (iv) for all non-leaf Selector gadget vertices at least half their neighbors are leaves (which are on the opposite side of the partition) (v) all Propagator vertices have exactly $n+1$ neighbors on each side of the partition. The total number of edges cut so far is (i) L_1 edges incident on leaves (ii) L_2 edges incident on degree 2 vertices that belong to heavy edges (iii) one internal edge of each path $P_{(i,j)}$ giving km edges in total (iv) half of the $2n+2$ edges incident on each Propagator vertex, of which there are $k(m-1)$, giving $k(m-1)(n+1)$ in total. Summing up, we have already cut $L_1+L_2+km+k(m-1)(n+1)$ edges, meaning we can still cut $m(2n+6)$ edges. We will describe a stable partition of the Checker gadgets which cuts exactly $2n+6$ edges per gadget (not counting edges incident on leaves, since these are already counted in L_1), and since we have m Checker gadgets this will complete the proof.

Consider now the Checker gadget for edge j which connects $v_{(i_1,j_1)}$ to $v_{(i_2,j_2)}$ and without loss of generality assume that j is odd (otherwise the proof is identical with the roles of 0 and 1 reversed). We claim that one of the vertices $t^1_j, t^2_j, t^3_j, t^4_j$ must have neighbors on both sides of the partition in the Selector gadgets. To see this, suppose for contradiction that each of these vertices only has neighbors on one side of the partition so far. Then, since t^1_j is connected to $P^1_{(i_1,j)}$, which has color 0 and t^2_j is connected to $P^{n+1}_{(i_1,j)}$, which has color 1, and t^1_j is connected to the first j_1 vertices of $P_{(i_1,j)}$, we conclude that $\sigma(i_1)=j_1$, because the number of vertices of the path $P_{(i_1,j)}$ which have value 0 is $\sigma(i_1)$. With the same argument, we must have $\sigma(i_2)=j_2$, contradicting the hypothesis that σ encodes an independent set.

We can therefore assume that one of $t^1_j, t^2_j, t^3_j, t^4_j$ has neighbors on both sides of the partition in the Selector gadgets. Without loss of generality suppose that t^1_j has this property (the proof is symmetric in other cases). We complete the partition as follows: we assign values to T^2_j, T^3_j, T^4_j in a way that t^2_j, t^3_j, t^4_j have the same number of neighbors on each side of the partition and that both neighbors of b_j in T^2_j, T^4_j have value 0. This is always possible as t^2_j, t^4_j have a neighbor with value 1 in the Selectors, namely $P^{n+1}_{(i_1,j)}$ and $P^{n+1}_{(i_2,j)}$. We assign colors to T^1_j in a way that t^1_j has the same number of neighbors on each side and a_j has two neighbors with distinct values in $T^1_j \cup T^3_j$. This is always possible as we need to use both values in T^1_j, because t^1_j has neighbors with both values in $P_{(i_1,j)}$. We give b_j value 1, c_j value 1 and a_j value 0. This is stable as b_j has two neighbors of value 0, c_j has neighbors with distinct values, and a_j has two neighbors with value 1. Furthermore, vertices in $T^1_j \cup T^2_j \cup T^3_j \cup T^4_j$ are stable because half their neighbors are leaves which are on the other side of the partition, and the neighborhoods of $t^1_j, t^2_j, t^3_j, t^4_j$ are completely balanced, so these vertices can be arbitrarily set. The number of edges cut is half of the edges incident on $t^1_j, t^2_j, t^3_j, t^4_j$, giving $2n+2$ edges, plus two edges incident on each of a_j, b_j, giving a total of $2n+6$ edges. ◀

A.9 Proof of Lemma 15

Proof. Suppose we have a stable cut of size at most B. This cut must include all L_1 edges incident on leaves, and at least one edge for each of the L_2 degree two vertices which belong to heavy edges. Furthermore, if there is a heavy edge such that both of its endpoints have the same value, the number of edges cut incident on vertices that belong to heavy edges will be at least $L_2 + A$. However, $A = n^5 > km + k(m-1)(n+1) + m(2n+6)$, so we would have a cut of size strictly larger than B. We conclude that in all heavy edges the two endpoints have distinct values. Without loss of generality assume value 0 is given to p_0 and 1 to p_1.

We now observe that:

1. At least one internal edge of each path $P_{(i,j)}$ is cut.
2. At least $n+1$ edges incident on each Propagator vertex are cut.
3. At least $2n+6$ edges not incident to leaves are cut inside each Checker gadget.

For the first claim, observe that if the endpoints of heavy edges take distinct values, this implies that in each path $P_{(i,j)}$ the first and last vertex have distinct values, so at least one edge of the path must be cut. The second claim is based on the fact that Propagator vertices have degree $2n+2$. For the third claim, observe that $t_j^1, t_j^2, t_j^3, t_j^4$ have $4n+4$ edges incident on them, so at least $2n+2$ of these must be cut in a stable solution. Furthermore, a_j, b_j have degree 3, so at least 2 edges incident on each of these vertices are cut, giving a total of $2n+6$. (Here, we used the fact that $\{t_j^1, t_j^2, t_j^3, t_j^4, a_j, b_j\}$ is an independent set).

By the above observations we have that any stable cut must have size at least $L_1 + L_2 + km + k(m-1)(n+1) + m(2n+6) = B$. Furthermore, if a solution cuts more than one edge of a path $P_{(i,j)}$, or at least $n+2$ edges incident on a Propagator, or at least $2n+7$ edges not incident to leaves in a Checker, then its total size must be strictly larger than B. We conclude that our solution must cut exactly one edge inside each Selector, properly balance the neighborhoods of all Propagators, and cut $2n+6$ edges inside each Checker.

Consider now two consecutive Selector gadgets $P_{(i,j)}$ and $P_{(i,j+1)}$. Since the solution cuts exactly one internal edge of each path, we can assume that the first x vertices of $P_{(i,j)}$ have the same value as $P_{(i,j)}^1$ and the remaining $n+1-x$ have the same value as $P_{(i,j)}^{n+1}$. Similarly, the first y vertices of $P_{(i,j+1)}$ have the same value as $P_{(i,j+1)}^1$. Now, because $j, j+1$ have different parities, this means that the Propagator connected to these two paths has $n+1-x+y$ neighbors on the same side as $P_{(i,j)}^{n+1}$. But this implies that $x = y$. Using the same reasoning we conclude that for all i, j, j', the number of vertices of $P_{(i,j)}$ that share the value of $P_{(i,j)}^1$ is equal to the number of vertices of $P_{(i,j')}$ that share the value of $P_{(i,j')}^1$. Let $\sigma(i)$ be the number of vertices of $P_{(i,1)}$ which share the value of $P_{(i,1)}^1$. We claim that $\{v_{(i,\sigma(i))} \mid i \in [k]\}$ is an independent set in G.

To see this, suppose for contradiction that the j-th edge of G connects $v_{(i_1,\sigma(i_1))}$ to $v_{(i_2,\sigma(i_2))}$. We claim that in this case the Checker connected to $P_{(i_1,j)}, P_{(i_2,j)}$ will have at least $2n+7$ cut edges. Indeed, observe that in this case the neighborhoods of $t_j^1, t_j^2, t_j^3, t_j^4$ are all contained on one of the two sides of the partition. Then, either the neighborhood of one of these four vertices is not completely balanced, in which case the cut includes at least $2n+3$ edges incident on these plus at least 4 edges incident on a_j, b_j; or the sets $T_j^1, T_j^2, T_j^3, T_j^4$ are also all contained on one of the two sides of the partition and furthermore, $T_j^1 \cup T_j^3$ are on one side and $T_j^2 \cup T_j^4$ are on the other. This implies that a_j, b_j must be on distinct sides of the partition. As a result, no matter where c_j is placed, one of a_j, b_j will have all three of its incident edges cut and as a result at least $2n+7$ edges will be cut in this Checker. We conclude that σ must encode an independent set. ◀

A.10 Proof of Lemma 16

Proof. We will use the fact that deleting a vertex from a graph can decrease the pathwidth by at most 1, since we can take a path decomposition of the resulting graph and add this vertex to all bags. We begin by deleting p_0, p_1 from the graph, as this decreases the pathwidth by at most 2. We will also use the fact that deleting all leaves from a graph can decrease pathwidth by at most 1, since we can take a path decomposition of the resulting graph and, for each leaf, find a bag of this decomposition that contains the leaf's neighbor and insert a copy of this bag immediately after it, adding the leaf. We therefore remove all leaves from the graph, decreasing the pathwidth by at most 1 more. Let H be the resulting graph. We will show that H has pathwidth at most $O(k)$. Observe that in H all heavy edges have disappeared, as their internal vertices became leaves when we deleted p_0, p_1.

For $j \in [m]$ let H_j be the graph induced by the set that contains all vertices of H from Selector gadgets $P_{(i,j)}$ for $i \in [k]$, the (at most $2k$) Propagator vertices connected to them, and the Checker gadget for the j-th edge. We will construct a path decomposition of H_j with the property that all bags include all Propagator vertices of H_j. If we achieve this then we can make a path decomposition of H by gluing together these decompositions, connecting the last bag of the decomposition of H_j with the first bag of the decomposition of H_{j+1}. Observe that the union of the graphs H_j covers all vertices and edges of H.

To build such a path decomposition of H_j we can remove the $2k$ Propagators contained in H_j (since we will add them in all bags) and the vertices $t_j^1, t_j^2, t_j^3, t_j^4, a_j, b_j$, decreasing pathwidth by at most $2k+6$. But the resulting graph is a union of paths and isolated vertices, so has pathwidth 1. We can therefore build a decomposition of H_j – and by extension of H – of width $2k + O(1)$. ◀