On the Parameterized Complexity of Computing Graph Bisections[*]

René van Bevern[1], Andreas Emil Feldmann[2], Manuel Sorge[1], and Ondřej Suchý[3]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany[**]
{rene.vanbevern,manuel.sorge}@tu-berlin.de
[2] Combinatorics & Optimization, University of Waterloo, Canada
andreas.feldmann@uwaterloo.ca
[3] Faculty of Information Technology, Czech Technical University in Prague, Czech Republic[***] ondrej.suchy@fit.cvut.cz

Abstract. The BISECTION problem asks for a partition of the vertices of a graph into two equally sized sets, while minimizing the *cut size*. This is the number of edges connecting the two vertex sets. BISECTION has been thoroughly studied in the past. However, only few results have been published that consider the parameterized complexity of this problem.

We show that BISECTION is FPT w.r.t. the minimum cut size if there is an optimum bisection that cuts into a given constant number of connected components. Our algorithm applies to the more general BALANCED BISEPARATOR problem where vertices need to be removed instead of edges. We prove that this problem is W[1]-hard w.r.t. the minimum cut size and the number of cut out components.

For BISECTION we further show that no polynomial-size kernels exist for the cut size parameter. In fact, we show this for all parameters that are polynomial in the input size and that do not increase when taking disjoint unions of graphs. We prove fixed-parameter tractability for the distance to constant cliquewidth if we are given the deletion set. This implies fixed-parameter algorithms for some well-studied parameters such as cluster vertex deletion number and feedback vertex set.

1 Introduction

We consider the NP-hard BISECTION problem for which the n vertices of a graph $G = (V, E)$ need to be partitioned into two sets A and B of size at most $\lceil n/2 \rceil$ each ((A, B) is a *bisection* of G). At the same time the *cut size* needs to be minimized. This is the number of edges connecting vertices in A with vertices in B. Throughout this paper it will be convenient to consider the decision problem corresponding to BISECTION, which is defined as follows.

[*] To appear in *Proceedings of the 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2013)*, Lübeck, Germany, June 2013. © Springer

[**] Supported by DFG project DAPA, NI 369/12.

[***] The main work was done while O. Suchý was at TU Berlin, supported by the DFG, research project AREG, NI 369/9.

Bisection
Input: A graph G and a positive integer k.
Question: Does G have a bisection with cut size at most k?

Bisection is of importance both in theory and practice, and has applications in divide-and-conquer algorithms [26], computer vision [25], and route planning [12]. We study Bisection from the point of view of parameterized complexity, and consider several parameters (Table 1) that naturally arise from the known results for Bisection. That is, we consider a given parameter p of an input instance and ask whether an algorithm with running time $f(p) \cdot n^{O(1)}$ exists that optimally solves the problem. Here $f(p)$ is a function that only depends on p. If there is such an algorithm we say that the problem is *fixed-parameter tractable* (or FPT for short) with respect to p.

Bisection has been thoroughly studied in the past. It is known that it is NP-hard in general [20] and the minimum cut size can be approximated within a factor of $O(\log n)$ [32]. Assuming the Unique Games Conjecture, no constant factor approximations exist [23]. For special graph classes such as trees [27] and solid grids [17] the optimum cut size can be computed in polynomial time. For planar graphs it is still open whether Bisection is NP-hard, but it is known to be FPT with respect to the cut size [8].

In this paper we show that for general graphs one can find an optimal bisection in FPT-time with respect to the cut size if there is an optimal bisection that cuts the graph into a given constant number of connected components. This result is motivated by the fact that in practice the solutions are typically cut into very few connected components [2]. Also for random regular graphs the sets A and B of the optimum bisection are connected with high probability [9]. Our algorithm is presented for the more general Balanced Biseparator problem, in which vertices instead of edges need to be removed in order to bisect the graph. To achieve our result, we generalize the *treewidth reduction* technique for separation problems that has been recently introduced by Marx et al. [29]. By adapting it to the global balancedness constraint, we address an open question by Marx et al. [30] of whether this is possible. We furthermore observe that Balanced Biseparator is W[1]-hard with respect to the cut size and the number of cut out components. Hence, Balanced Biseparator is unlikely to be FPT even when combining these parameters. This means that to obtain a fixed-parameter algorithm it is unavoidable to impose some additional constraint. We chose our condition on having a constant number of cut out connected components in the optimum solution as a natural candidate, as argued above.

Whether Bisection is FPT with respect to the cut size alone, though, remains open. However, we show that no polynomial-size problem kernels exist for this parameter, unless coNP $\subseteq$ NP/poly. Hence, it is unlikely that there is a polynomial-time algorithm that computes an instance of size polynomial in the cut size and equivalent to the original instance. We prove this by giving a corresponding result for all parameters that are polynomial in the input size and that do not increase when taking the disjoint union of graphs. This includes parameters such as treewidth, minimum cut size, cliquewidth, and more generally bandwidth (see [5, Theorem 44]).

Table 1. Overview of known and new parameterized results.

Parameter	Results for Bisection
cut size	FPT for planar graphs [8]
	FPT for constant cut out components (Theorem 2)
	No poly-size kernel (Theorem 3)
	W[1]-hard for Balanced Biseparator (deferred)
treewidth	FPT [33, 34]
	No poly-size kernel (Theorem 3)
cliquewidth	XP (Lemma 3)
	No poly-size kernel (Theorem 3)
bandwidth	No poly-size kernel (Theorem 3)
feedback vertex set	FPT (Corollary 1)
cluster vertex deletion number	FPT (Corollary 1)

Some of these parameters have been considered for the Bisection problem before. For instance, we already mentioned the cut size, and it was shown that the problem is FPT with respect to treewidth [33, 34]. However, although treewidth is probably the most widely used graph parameter for sparse graphs, it is not suitable for dense graphs, although they can also have simple structure. For that purpose, Courcelle and Olariu [11] introduced the parameter *cliquewidth* [14]. For this parameter we present an XP algorithm, i.e. an algorithm finding the optimum solution in time $n^{O(q)}$ if a cliquewidth-q expression for the graph is given. In fact we obtain an algorithm that shows that Bisection is FPT with respect to the *cliquewidth-q vertex deletion number*:[4] the number of vertices that has to be deleted to obtain a graph of constant cliquewidth q. To the best of our knowledge this parameter has not been considered in the past. The cliquewidth-q deletion number is a generalization of several well-studied graph parameters like vertex cover ($q = 1$) [10], cluster vertex deletion number and cograph vertex deletion number ($q = 2$) [11], or feedback vertex set ($q = 3$) [24] and treewidth-t vertex deletion set ($q = 2^{t+1} + 1$) [11, 18].

In this paper we use standard terminology of graph theory [13]. Due to space constraints, many proofs are deferred to the full version of the paper.

2 An FPT Algorithm for Cut Size and Constant Number of Cut Out Components

This section shows that Bisection is FPT with respect to the cut size if there is an optimum bisection that cuts into at most some given constant number of connected components. To this end, we show an FPT-algorithm for the more general problem Balanced Biseparator; to formally define it, we need some terminology. Let G be a graph and $S \subseteq V(G)$. We call S an *A-B-separator* if there are vertex sets $A, B \subseteq V(G)$ such that S, A, B form a partition of $V(G)$, and there

[4] Precisely, we need the vertex deletion set to be given to obtain FPT for this parameter.

are no edges between A and B in G. Moreover, we call S *balanced* if $||A|-|B|| \leq 1$. We say that G has a *balanced separator* S if there are sets A, B such that S is a balanced A-B-separator for G. We say that S is an s-t-separator for vertices s, t if there are vertex sets $A, B \subseteq V(G)$ such that S is an A-B-separator and $s \in A$ and $t \in B$. We say that an s-t-separator S is *inclusion-wise minimal*, or just *minimal*, if there is no s-t-separator $S' \subsetneq S$. Finally, we say that S is a c-component separator for G, if there are c connected components in $G - S$. Balanced Biseparator is the following problem:

Balanced Biseparator
Input: A graph G and a positive integer k.
Question: Does G have a balanced separator of size at most k?

Using a reduction from the W[1]-hard problem Cutting ℓ Vertices [28], one can show that Balanced Biseparator is W[1]-hard with respect to k and the number c of cut out components. Hence, an additional constraint like c being constant is unavoidable to get an FPT-algorithm. Our algorithm for Balanced Biseparator transfers also to Bisection:

Proposition 1. *There is a polynomial-time many-one reduction from* Bisection *to* Balanced Biseparator *such that the desired separator size is at most one larger than the desired cut size. Furthermore, each bisection with* c *connected components for the* Bisection *instance yields a balanced separator for the* Balanced Biseparator *instance whose removal leaves at most* $c + 2$ *connected components and vice-versa.*

Proposition 1 implies that Balanced Biseparator is a more general problem than Bisection. We now outline an FPT algorithm for Balanced Biseparator: we first observe that a balanced separator consists of minimal s-t-separators between a collection of "terminal" vertices s, t. The terminal vertices are chosen one from each of the connected components of the input graph without the separator. Guessing the terminals, we can reduce Balanced Biseparator to finding an "almost balanced" separator consisting of vertices contained in minimal separators. To find such a separator, we generalize the "treewidth reduction" technique introduced by Marx et al. [29, 30] to graphs with vertex weights. We obtain an algorithm that constructs a weighted graph G' that preserves all inclusion-wise minimal vertex cuts of size at most k between some given terminals, preserves the weight of the cut out parts, and has treewidth bounded by some function $g(k, c)$ where c is the number of terminals. Moreover the algorithm runs in time $f(k, c) \cdot n^{O(1)}$. We then show that Balanced Biseparator is fixed-parameter tractable with respect to treewidth when fixing the number of components of the separated graph. The final algorithm guesses the terminals, reduces the treewidth and then solves the bounded-treewidth problem.

The main ingredient in our FPT algorithm for Balanced Biseparator is a generalization of the treewidth reduction technique of Marx et al. [30] to graphs with vertex weights: we aim to construct a graph of bounded treewidth that preserves all inclusion-wise minimal s-t-separators of a given size. To this end, we define trimmers. Let $G = (V, E)$ be a graph, k an integer and $T \subseteq V$. A tuple (G^*, ϕ) of a graph $G^* = (V^*, E^*)$ and a total, surjective, but not necessarily injective mapping $\phi\colon V \to V^*$ is called a (k, T)-*trimmer* of G if the following holds. (Here, we extend $\phi(V) := \bigcup_{v \in V} \phi(v)$ and $\phi^{-1}(v) := \{v' \mid \phi(v') = v\}$.)

(i) Let $S \subseteq V^*$. A set $C \subseteq V^*$ is a connected component in $G^* - S$ if and only if $\phi^{-1}(C)$ is a connected component in $G - \phi^{-1}(S)$ (i. e., ϕ implies a one-to-one mapping between the connected components of $G^* - S$ and $G - \phi^{-1}(S)$).

(ii) If S is an inclusion-wise minimal s-t-separator for G with $|S| \le k$ and $s, t \in T$, then $\phi(S) = S$ and $\phi(S)$ is an inclusion-wise minimal $\phi(s)$-$\phi(t)$-separator for G^*.

We obtain the following.

Theorem 1. *Let $G = (V, E)$ be a graph. For every constant $k \in \mathbb{N}$ and constant-size $T \subseteq V$, we can compute a (k, T)-trimmer (G^*, ϕ) for G in $O(|V| + |E|)$ time such that the treewidth of G^* is at most $g(k, |T|)$ for some function g depending only on k and $|T|$. Moreover, both ϕ and ϕ^{-1} are linear-time computable with respect to their output length.*

We can now state an algorithm for finding a c-component balanced separator of a given size. To this end, we first note that Balanced Biseparator is FPT with respect to treewidth and, thus, gathering the final ingredient for the algorithm.

Lemma 1. *Let G be a graph with treewidth ω and integer vertex-weights λ. Let Λ be the sum of all vertex weights and let $c \ge 2$ be an integer. We can find in $\omega^{O(\omega)} \cdot c^2 \cdot \Lambda^2 \cdot n$ time, for all integers $1 \le s \le \Lambda$, a minimum weight c-component A-B-separator S with $\lambda(A) = s$, or reveal that no such separator exists.*

We now arrive at the main theorem of this section.

Theorem 2. *Let G be a graph. Given non-negative integers c and k, in $h(c, k) \cdot n^{c+3}$ time we can find a c-component balanced separator for G of size at most k if it exists. Here, $h(c, k)$ is a function depending only on c and k.*

Proof. The algorithm proceeds as follows. For each $T \subseteq V(G)$ of size c we compute a (k, T)-trimmer (G^*, ϕ) using Theorem 1. We create a vertex weight function λ for G^* by letting $\lambda(v) = |\phi^{-1}(v)|$. Then, for each s, $|V(G)|/2 - 1 - k \le s \le |V(G)|/2 + k$, we compute a minimum-weight c-component A'-B'-separator for G^* with $\lambda(A') = s$ using Lemma 1. If among these separators there is an A'-B'-separator S' with $|\lambda(A') - \lambda(B')| \le k - \lambda(S') + 1$, then we compute $S := \phi^{-1}(S')$, $A := \phi^{-1}(A')$, and $B := \phi^{-1}(B')$. Note that, by trimmer property (i), S is a c-component A-B-separator for G. Moreover, since ϕ is a total mapping, $||A| - |B|| \le k - |S| + 1$. We move $k - |S|$ vertices from A or B to S such that S is a c-component balanced separator for G and we output S. Note that, unless $|V(G)|$ is bounded by a function of k and the problem is trivial, moving the vertices to S without changing the number of components is always possible, because not every vertex of a connected component can separate it into multiple ones and there is always a component of size at least two. If no suitable separator is found, we output that there is no c-component balanced separator of size at most k for G.

Let S be a c-component balanced separator of size k for G and pick vertices $v_1, \ldots, v_c$, one from each connected component of $G-S$. Let us observe that the above algorithm finds a c-component balanced separator of size at most k. Note that S is a v_i-v_j-separator for each $1 \leq i < j \leq c$. Hence, S contains inclusion-wise minimal v_i-v_j-separators $S_{i,j}$ of size at most k. Let $\hat{S} = \bigcup_{1 \leq i < j \leq c} S_{i,j}$, call a connected component in $G - \hat{S}$ *odd* if it does not contain any v_i, and let $\tilde{S}$ be the union of $\hat{S}$ and all odd components. Note that odd components are contained in S. Hence, $\tilde{S}$ is a c-component $\tilde{A}$-$\tilde{B}$-separator for G with $||\tilde{A}|-|\tilde{B}|| \leq k-|\tilde{S}|+1$ and $|V(G)|/2-1-k \leq |\tilde{A}| \leq |V(G)|/2+k$. By trimmer property (ii) we have that $\phi(\hat{S}) = \hat{S}$ is contained in G^*. Thus, by trimmer property (i), ϕ implies a one-to-one mapping of connected components C in $G-\hat{S}$ and their counterparts $\phi(C)$ in $G^* - \hat{S}$. In particular, there is such a mapping for all odd connected components. Thus, $\phi(\tilde{S})$ is a c-component $\phi(\tilde{A})$-$\phi(\tilde{B})$-separator for G^* and we have $\lambda(\phi(\tilde{S})) = |\tilde{S}|, \lambda(\phi(\tilde{A})) = |\tilde{A}|$, and $\lambda(\phi(\tilde{S})) = |\tilde{S}|$. Hence, an A'-B'-separator S' for G^* with $\lambda(S') \leq \lambda(\phi(\tilde{S}))$ and $\lambda(A') = \lambda(\phi(\tilde{A}))$ is enumerated by the algorithm of Lemma 1. Applying the size bounds of $\tilde{A}, \tilde{B}, \tilde{S}$ we have $||\lambda(A')|-|\lambda(B')|| \leq k-|\lambda(S')|+1$ and $|V(G)|/2-1-k \leq |\lambda(A')| \leq |V(G)|/2+k$. Thus, the algorithm described above finds a c-component balanced separator of size at most k for G. The proof of the running time bound is deferred to a full version of the paper. $\square$

3 Incompressibility

Problem kernelization is a powerful preprocessing tool in attacking NP-hard problems [6, 21]. A *reduction to a problem kernel* is an algorithm that, given an instance I with parameter p of a parameterized problem, in time polynomial in $(|I|+p)$ outputs an instance I' of the same problem and a parameter p' such that

i) I is a yes-instance if and only if I' is a yes-instance,
ii) $|I'|+p' \leq f(p)$, where f is a function only depending on p.

The function f is called the *size* of the problem kernel. It is desirable to find problem kernels of size polynomial in the parameter p.

In this section, we show that Bisection has no polynomial-size kernel with respect to any parameter that is polynomial in the input size and does not increase when taking disjoint unions of graphs. Our result excludes polynomial-size problem kernels for the parameters treewidth, cut size of the bisection (the "standard parameter"), cliquewidth, and more generally pathwidth (see [5, Theorem 44]).

Theorem 3. *Unless coNP ⊆ NP/poly,* Bisection *does not have polynomial-size kernels with respect to any parameter that is polynomial in the input size and that does not increase when taking disjoint unions of graphs.*

For Theorem 3, we first show that a version of Bisection with integer edge weights does not have a polynomial-size kernel, and then show how to remove the weights. To prove that Edge-Weighted Bisection does not have a polynomial-size kernel, it is sufficient to show a cross composition (cf. Bodlaender et al. [7]) from the NP-hard [19] Maximum Cut problem to Edge-Weighted Bisection.

Maximum Cut
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a partition of V into sets A and B such that at least k edges have one endpoint in A and one in B?

Lemma 2. *There is a cross composition of* Maximum Cut *to* Edge-Weighted Bisection *with respect to any parameter that is polynomial in the input size and does not increase when taking the disjoint union of graphs.*

Showing the cross composition amounts to the following. We give a polynomial-time algorithm that transforms input instances $(G_1, k_1), \dots, (G_t, k_t)$ of Maximum Cut into one instance (G^*, k^*) of Edge-Weighted Bisection such that (G^*, k^*) is a yes-instance if and only if one of the Maximum Cut instances is and such that k^* is polynomial in the size of the largest input instance.

Construction 1. The construction resembles the reduction given for the NP-hardness of Bisection by Garey et al. [20]. To easier present the construction, without loss of generality we assume that

i) each of the G_i, $1 \leq i \leq t$, has exactly n vertices and $k_1 = \dots = k_t =: k$ (cf. Bodlaender et al. [7]),
ii) $1 \leq k \leq n^2$: if $k = 0$, all instances are yes-instances, and if $k > n^2$, all instances are no-instances. Hence, if not $1 \leq k \leq n^2$, we can return a trivial yes-instance or no-instance of Edge-Weighted Bisection,
iii) t is odd: otherwise, we can add a no-instance to the list of input instances that consists of the edgeless graph on n vertices.

Since the output graph G^* will consist of connected components, each having at most $2n$ vertices, and since our parameter is polynomial in the input size and does not increase when taking the disjoint union of graphs, we trivially obtain that the output parameter is polynomial in n. We create G^* as follows: for each input graph $G_i = (V_i, E_i)$, $1 \leq i \leq t$, add to G^* the vertices in V_i and a clique V_i' with $|V_i|$ vertices and edges of weight $W := n^2$ each. All vertices in V_i' are adjacent to all vertices in V_i in G^* via an edge of weight W. Now, for each pair $v, w \in V_i$, add an edge $\{v, w\}$ to G^* with weight W if $\{v, w\} \notin E_i$ and with weight $W - 1$ if $\{v, w\} \in E_i$. We set $k^* := Wn^2 - k$.

We use Construction 1 to show Lemma 2 and subsequently Theorem 3.

4 FPT for the Cliquewidth-q Vertex Deletion Number

This section shows Bisection to be fixed-parameter tractable with respect to the number of vertices that have to be removed from a graph to reduce its cliquewidth to a constant q. Thus, we generalize many well-studied graph parameters like vertex cover ($q = 1$) [10], cluster vertex deletion number and cograph vertex deletion number ($q = 2$) [11], or feedback vertex set ($q = 3$) [24] and treewidth-t vertex deletion set [18]. Our definition of cliquewidth is inspired by Hliněný et al. [22].

Let q be a positive integer. We call (G, λ) a *q-labeled graph* if G is a graph and $\lambda : V(G) \to \{1, 2, \dots, q\}$ is a mapping. The number $\lambda(v)$ is called *label* of a vertex v. We introduce the following operations on labeled graphs:

(1) For every i in $\{1, \ldots, q\}$, we let $\bullet_i$ denote the graph with only one vertex that is labeled by i (a constant operation).
(2) For every pair of distinct $i, j \in \{1, 2, \ldots, q\}$, we define a unary operator $\eta_{i,j}$ such that $\eta_{i,j}(G, \lambda) = (G', \lambda)$, where $V(G') = V(G)$, and $E(G') = E(G) \cup \{(v, w) \mid v, w \in V, \lambda(v) = i, \lambda(w) = j\}$. In other words, the operator adds all edges between label-i vertices and label-j vertices.
(3) For every pair of distinct $i, j \in \{1, 2, \ldots, q\}$, we let $\rho_{i \to j}$ be the unary operator such that $\rho_{i \to j}(G, \lambda) = (G, \lambda')$, where $\lambda'(v) = j$ if $\lambda(v) = i$, and $\lambda'(v) = \lambda(v)$ otherwise. The operator only changes the labels of vertices labeled i to j.
(4) Finally, $\oplus$ is a binary operation that makes the disjoint union, while keeping the labels of the vertices unchanged. Note explicitly that the union is disjoint in the sense that $(G, \lambda) \oplus (G, \lambda)$ has twice the number of vertices of G.

A *q-expression* is a well-formed expression φ written with these symbols. The q-labeled graph produced by performing these operations therefore has a vertex for each occurrence of the constant symbol in φ; and this q-labeled graph (and any q-labeled graph isomorphic to it) is called the *value* $val(\varphi)$ of φ. If a q-expression φ has value (G, λ), we say that φ is a *q-expression of* G. The *clique-width* of a graph G, denoted by $cwd(G)$, is the minimum q such that there is a q-expression of G. We say that a join $\eta_{i,j}$ is *full* if there is no edge between vertices of label i and j in the labeled graph on which the join is applied.

Proposition 2. *For any q-expression for an n-vertex graph there is an equivalent one which is at most as long as φ, contains $O(q^2 \cdot n)$ symbols, and for which every join is full.*

In the following, we show how to compute an optimal bisection using the q-expression of a given graph G. This will naturally also solve the decision problem Bisection. Let $D \subseteq V(G)$ and φ be a q-expression for $G \setminus D$, i.e. $val(\varphi) = (G \setminus D, \lambda)$. Let A_0, B_0 be a partition of D. For now, we assume that there are no edges between A_0 and B_0. Let $n_i(\varphi)$ for $i \in \{1, \ldots, q\}$ be the number of vertices of $G \setminus D$ with label i. For every pair of vectors $\boldsymbol{a} = (a_1, \ldots, a_q), \boldsymbol{b} = (b_1, \ldots, b_q) \in \mathbb{N}^q$ with $a_i + b_i = n_i(\varphi)$, let us denote by $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$ the minimum number of edges between different parts of a partition (A, B) of $V(G)$ which satisfies the following conditions: (i) $A_0 \subseteq A$, $B_0 \subseteq B$, (ii) the number of vertices in $A \setminus D$ and $B \setminus D$ of label i are a_i and b_i, respectively. In the following we use x_i to denote the i'th entry in a vector $\boldsymbol{x}$.

Lemma 3. *There is an algorithm that for given G, A_0, B_0 and φ in time $O(n^{2q} \cdot q \cdot |\varphi|)$ computes all the numbers $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$.*

Proof. We prove the lemma by induction on the length of the q-expression. By Proposition 2 we can assume that every join in φ is full. If $\varphi = \bullet_i$, then we have $n_i(\varphi) = 1$ and $n_j(\varphi) = 0$ for every $j \neq i$. Hence, in each pair of q-dimensional vectors $\boldsymbol{a}, \boldsymbol{b}$ there is either $a_i = 1$ or $b_i = 1$ and the other numbers are zero. In this case, there is exactly one partition fulfilling the conditions (i) and (ii), namely the one which puts the only vertex of $G \setminus D$ to set A or B as required. It is easy to compute the number of edges between the parts in this partition.

Now, suppose $\varphi = \eta_{i,j}(\varphi')$. Since φ' is shorter than φ, by the induction hypothesis we can use an algorithm for φ' to store all the results in a table $Cut_{A_0,B_0}(\varphi', \boldsymbol{a}, \boldsymbol{b})$. Note that $val(\varphi')$ differs from $G \setminus D$ only in that $G \setminus D$ has an edge between every vertex of label i and every vertex of label j, while $val(\varphi')$ has no such edges (as the join is full). Therefore, every partition (A, B) of $G \setminus D$ fulfilling the conditions (i) and (ii), is also a partition for $val(\varphi')$ fulfilling these conditions, but in $G \setminus D$ there are exactly $a_i \cdot b_j + a_j \cdot b_i$ more edges between the parts. Hence, we can output $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b}) = Cut_{A_0,B_0}(\varphi', \boldsymbol{a}, \boldsymbol{b}) + a_i \cdot b_j + a_j \cdot b_i$.

Next, let us assume that $\varphi = \rho_{i \to j}(\varphi')$, and the table containing the values of $Cut_{A_0,B_0}(\varphi', \boldsymbol{a}', \boldsymbol{b}')$ is already computed. Note that in $G \setminus D$ there are no vertices of label i, so we have $0 = n_i(\varphi) = a_i = b_i$. On the other hand, some of the vertices which have label j in $G \setminus D$ had label i in $val(\varphi')$. A minimal partition for $G \setminus D$, $\boldsymbol{a}$, and $\boldsymbol{b}$ which satisfies the conditions (i) and (ii) is also a partition for $val(\varphi')$ which satisfies the conditions (i) and (ii) for some $\boldsymbol{a}', \boldsymbol{b}'$, but we don't know the distributions of a_j to a'_j and a'_i and of b_j to b'_j and b'_i. Therefore $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$ can be computed as $\min\{Cut_{A_0,B_0}(\varphi', \boldsymbol{a}', \boldsymbol{b}')\}$ where the minimum is taken over all pairs $\boldsymbol{a}', \boldsymbol{b}'$ where $a'_t = a_t$ and $b'_t = b_t$ for every $t \in \{1, \dots, q\} \setminus \{i, j\}$; $a_j = a'_j + a'_i$; $b_j = b'_j + b'_i$ and $a'_t + b'_t = n_t(\varphi')$ for $t \in \{i, j\}$. As every pair $\boldsymbol{a}', \boldsymbol{b}'$ gives rise to exactly one $\boldsymbol{a}, \boldsymbol{b}$, all the minima can be computed in one pass over all $\boldsymbol{a}', \boldsymbol{b}'$.

Finally, let $\varphi = \varphi^1 \oplus \varphi^2$ and let the values of $Cut_{A_0,B_0}(\varphi^1, \boldsymbol{a}^1, \boldsymbol{b}^1)$ and $Cut_{A_0,B_0}(\varphi^2, \boldsymbol{a}^2, \boldsymbol{b}^2)$ be already computed and stored in a table. A minimal partition for $G \setminus D$ and $\boldsymbol{a}, \boldsymbol{b}$ satisfying the conditions (i) and (ii) also induces partitions for $val(\varphi^1)$ and $val(\varphi^2)$, which satisfy the conditions (i) and (ii) for some $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$, but we don't know the distributions of a_i to a^1_i and a^2_i and of b_i to b^1_i and b^2_i. Moreover, there are no edges between $val(\varphi^1)$ and $val(\varphi^2)$. Thus $\min\{Cut_{A_0,B_0}(\varphi^1, \boldsymbol{a}^1, \boldsymbol{b}^1) + Cut_{A_0,B_0}(\varphi^2, \boldsymbol{a}^2, \boldsymbol{b}^2)\}$ gives $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$, where the minimum is taken over all $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$ where for every $i \in \{1, \dots, q\}$, $a_i = a^1_i + a^2_i$, $b_i = b^1_i + b^2_i$, and $a^l_i + b^l_i = n_i(\varphi^l)$ for $l \in \{1, 2\}$. As every pair of pairs $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$ gives rise to exactly one pair $\boldsymbol{a}, \boldsymbol{b}$, all the minima can be computed in one pass over all combinations of $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$.

Concerning the running time, we again argue by induction to show that the overall time is $O(n^{2q} \cdot q \cdot |\varphi|)$. If $\varphi = \bullet_i$, then $|\varphi| = 1$ and the computation of Cut_{A_0,B_0} for the only possible pair of q-dimensional vectors takes $O(m + n) \subseteq O(n^{2q} \cdot q)$ time. This constitutes the induction basis. Otherwise, for any sub-expression φ' of a given expression φ, the computation of the table for φ' takes $O(n^{2q} \cdot q \cdot |\varphi'|)$ time by the induction hypothesis. Observe that there are $O(n^q)$ different pairs of q-dimensional vectors $\boldsymbol{a}, \boldsymbol{b}$ with $a_i + b_i = n_i(\varphi)$. If $\varphi = \eta_{i,j}(\varphi')$, then the computation for each pair of vectors takes $O(q)$ time. For $\varphi = \rho_{i \to j}(\varphi')$, one pass through the table of φ' is obviously accomplished in $O(n^q)$ time, spending $O(1)$ time per entry. Since in both cases $|\varphi| = |\varphi'| + 1$, this proves the time bound for φ for these expressions. Finally, if $\varphi = \varphi^1 \oplus \varphi^2$ then the tables for φ^1 and φ^2 can be computed in $O(n^{2q} \cdot q \cdot (|\varphi^1| + |\varphi^2|))$ time. Then we cycle over the entries of both tables and for each combination we spend $O(q)$ time, so this can be accomplished in $O(n^{2q} \cdot q)$ time. Since $|\varphi| = |\varphi^1| + |\varphi^2| + 1$, also in this case the algorithm runs in $O(n^{2q} \cdot q \cdot |\varphi|)$ time. $\square$

Theorem 4. *For G a graph, $D \subseteq V(G)$, and φ a q-expression for $G \setminus D$ there is an $O(2^{|D|} \cdot n^{2q+1} q^3)$ time algorithm which computes the optimal bisection of G.*

Proof. It is enough to find the minimum of $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$ over all partitions A_0, B_0 and pairs of q-dimensional vectors $\boldsymbol{a}, \boldsymbol{b}$ with $|A_0| + \sum_{i=1}^{q} a_i$ equal to $|B_0| + \sum_{i=1}^{q} b_i$. Since Lemma 3 only applies when there are no edges between A_0 and B_0, we delete them and add the number of them to the sum. As the size of φ is $O(q^2 \cdot n)$ by Proposition 2, the running time follows from Lemma 3. $\square$

Given D, an $f(q)$-expression for $G \setminus D$ can be computed in polynomial time using a cliquewidth approximation [31]. Thus, Bisection is FPT with respect to the size of any constant-cliquewidth vertex-deletion set that is obtainable in FPT time.

Corollary 1. Bisection *is fixed-parameter tractable with respect to the size of a feedback vertex set, the size of a cluster vertex deletion set, and the size of a treewidth-t vertex deletion set.*

It is easy to generalize Theorem 4 to Balanced d-Partitioning, where one searches for a partition into some constant $d > 2$ many equal-sized parts. The running time achieved is $O(d^{|D|+1} \cdot n^{2(d-1)q+1} q^3)$. We note that such a running time bound is tight in the sense that there is no algorithm with running time $f(d, |D|) n^{O(1)}$ for constant q unless FPT = W[1]: since the deletion of a feedback vertex set leaves a forest, the resulting graph has clique-width at most 3 [11]. Thus, if there was an algorithm with the above running time, then Balanced Partitioning would be fixed-parameter tractable with respect to the combined parameter size of a minimum feedback vertex set and number of parts in the partition. However, we can show that this parameter combination yields a W[1]-hard problem.

5 Conclusion

A natural generalization of the Bisection problem is to partition the graph into d equally-sized sets, for some arbitrary d instead of only two. This problem is called Balanced Partitioning and is considerably harder than Bisection. For instance Balanced Partitioning is hard to approximate even on trees [15]. Nonetheless it is of great importance in applications such as parallel computing [3] and VLSI circuit design [4]. Due to the hardness results [15] it was asked whether the problem is FPT for parameters resulting in algorithms useful in practice. Many of the known results already rule out FPT algorithms for some parameters such as treewidth or cluster vertex deletion number (Balanced Partitioning is NP-hard for trees [16] and graphs formed by a disjoint union of cliques [1]). We addressed this question and were able to show that the problem is W[1]-hard for the *combined* parameter cut size, feedback vertex set, treewidth, and number d of partitions.[5] We can, however, show that Balanced Partitioning is FPT with respect to the vertex cover number.[5]

[5] These results are deferred to a full version.

The main open problem remaining from this paper is the status of the parameterized complexity of Bisection with respect to the parameter cut size alone. But also, for the Balanced Partitioning problem, the question posed by Feldmann [15] of whether practical algorithms beyond the standard deterministic worst-case scenario exist, remains unanswered.

Acknowledgments. We thank Bart M. P. Jansen, Stefan Kratsch, Rolf Niedermeier and the anonymous referees for helpful suggestions. We are furthermore grateful to Rolf Niedermeier for many interesting discussions.

Bibliography

[1] K. Andreev and H. Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.
[2] P. Arbenz. Personal communication, 2013. ETH Zürich.
[3] P. Arbenz, G. van Lenthe, U. Mennel, R. Müller, and M. Sala. Multi-level μ-finite element analysis for human bone structures. In *Proc. 8th PARA*, volume 4699 of *LNCS*, pages 240–250. Springer, 2007.
[4] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *J. Comput. Syst. Sci.*, 28(2):300–343, 1984.
[5] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Science*, 209(1–2):1–45, 1998.
[6] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proc. 4th IWPEC*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009.
[7] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *Proc. 28th STACS*, volume 9 of *LIPIcs*, pages 165–176. Dagstuhl, 2011.
[8] T. N. Bui and A. Peck. Partitioning planar graphs. *SIAM J. Comput.*, 21 (2):203–215, 1992.
[9] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.
[10] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.
[11] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101(1-3):77–114, 2000.
[12] D. Delling, A. V. Goldberg, T. Pajor, and R. F. F. Werneck. Customizable route planning. In *Proc. 10th SEA*, volume 6630 of *LNCS*, pages 376–387. Springer, 2011.
[13] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.
[14] W. Espelage, F. Gurski, and E. Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *Proc. 27th WG*, volume 2204 of *LNCS*, pages 117–128. Springer, 2001.
[15] A. E. Feldmann. Fast balanced partitioning is hard, even on grids and trees. *Theor. Comput. Sci.*, 485:61–68, 2013.

[16] A. E. Feldmann and L. Foschini. Balanced partitions of trees and applications. In *Proc. 29th STACS*, volume 14 of *LIPIcs*, pages 100–111. Dagstuhl, 2012.

[17] A. E. Feldmann and P. Widmayer. An $O(n^4)$ time algorithm to compute the bisection width of solid grid graphs. In *Proc. 19th ESA*, volume 6942 of *LNCS*, pages 143–154. Springer, 2011.

[18] F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal fpt algorithms. In *Proc. 53rd FOCS*, pages 470–479. IEEE Computer Society, 2012.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., 1979.

[20] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Science*, 1(3):237–267, 1976.

[21] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

[22] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

[23] S. A. Khot and N. K. Vishnoi. The Unique Games Conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In *Proc. 46th FOCS*, pages 53–62. IEEE Computer Society, 2005.

[24] T. Kloks, C. M. Lee, and J. Liu. New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In *Proc. 28th WG*, volume 2573 of *LNCS*, pages 282–295. Springer, 2002.

[25] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM T. Graphic.*, 22(3): 277–286, 2003.

[26] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9:615–627, 1980.

[27] R. M. MacGregor. *On Partitioning a Graph: a Theoretical and Empirical Study.* PhD thesis, University of California, Berkeley, 1978.

[28] D. Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.

[29] D. Marx, B. O'Sullivan, and I. Razgon. Treewidth reduction for constrained separation and bipartization problems. In *Proc. 27th STACS*, volume 5 of *LIPIcs*, pages 561–572. Dagstuhl, 2010.

[30] D. Marx, B. O'Sullivan, and I. Razgon. Finding small separators in linear time via treewidth reduction. *CoRR*, abs/1110.4765, 2011.

[31] S. Oum. Approximating rank-width and clique-width quickly. *ACM T. Algorithms*, 5(1), 2008.

[32] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th STOC*, pages 255–264. ACM, 2008.

[33] K. Soumyanath and J. S. Deogun. On the bisection width of partial k-trees. In *Proc. 20th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, volume 74 of *Congressus Numerantium*, pages 25–37, 1990.

[34] M. Wiegers. The k-section of treewidth restricted graphs. In *Proc. 15th MFCS*, volume 452 of *LNCS*, pages 530–537. Springer, 1990.