The minimum k-way cut of bounded size is fixed-parameter tractable

Ken-ichi Kawarabayashi
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku
Tokyo 101-8430, Japan
k_keniti@nii.ac.jp

Mikkel Thorup
AT&T Labs—Research
180 Park Avenue, Florham Park, NJ 07932, USA
mthorup@research.att.com

Abstract— We consider the minimum k-way cut problem for unweighted undirected graphs with a size bound s on the number of cut edges allowed. Thus we seek to remove as few edges as possible so as to split a graph into k components, or report that this requires cutting more than s edges. We show that this problem is fixed-parameter tractable (FPT) with the standard parameterization in terms of the solution size s. More precisely, for $s = O(1)$, we present a quadratic time algorithm. Moreover, we present a much easier linear time algorithm for planar graphs and bounded genus graphs.

Our tractability result stands in contrast to known W[1] hardness of related problems. Without the size bound, Downey et al. [2003] proved that the minimum k-way cut problem is W[1] hard with parameter k, and this is even for simple unweighted graphs. Downey et al. asked about the status for planar graphs. We get linear time with fixed parameter k for simple planar graphs since the minimum k-way cut of a planar graph is of size at most $6k$. More generally, we get FPT with parameter k for any graph class with bounded average degree.

A simple reduction shows that vertex cuts are at least as hard as edge cuts, so the minimum k-way vertex cut is also W[1] hard with parameter k. Marx [2004] proved that finding a minimum k-way vertex cut of size s is also W[1] hard with parameter s. Marx asked about the FPT status with edge cuts, which we prove tractable here. We are not aware of any other cut problem where the vertex version is W[1] hard but the edge version is FPT, e.g., Marx [2004] proved that the k-terminal cut problem is FPT parameterized by the cut size, both for edge and vertex cuts.

1. Introduction

We consider the *minimum k-way cut problem*[1] of an undirected graph. The goal is to find a minimum set of *cut* edges so as to split the graph into at least k components. This paper is focused on unweighted graphs, so graphs are unweighted unless stated otherwise.

Finding a minimum k-way cut is an extension of the classical minimum cut problem, and it has applications in the area of VLSI system design, parallel computing systems, clustering, network reliability and finding cutting planes for the traveling salesman problem.

Goldschmidt and Hochbaum [11] proved that finding a minimum k-way cut is NP-hard when k is part of the input, but polynomial time solvable for fixed k. Their algorithm finds a minimum k-way cut in $O(n^{(1/2-o(1))k^2})$ time. Karger and Stein [17] proposed an extremely simple randomized Monto Carlo algorithm for the k-way cut problem whose running time is $O(n^{(2-o(1))k})$. Then Kamidoi et al. [15] presented a deterministic algorithm that runs in $O(n^{(4+o(1))k})$ time, and finally, Thorup [22] presented the current fastest deterministic algorithm with a running time of $\tilde{O}(mn^{2k-2})$. The above results all hold for both weighted and unweighted graphs.

For exact algorithms, the obvious next complexity target is to get fixed-parameter tractability [9]. Recall that a problem is fixed-parameter tractable with parameter t if there is an algorithm with running time $f(t)n^c$ for some fixed function f and constant c. In parameterized complexity, W[1]-hardness plays a role similar to NP-hardness versus polynomial time. If a problem is W[1] hard with parameter t, we take it as evidence that it is not FPT with parameter t (see [9] for details including the somewhat technical definition of W[1]). For unweighted minimization problems, the generic standard parameter is the solution size, which in our case is the cut size. With the standard parameterization, we get a nice relation to approximation schemes that we shall return to later.

Downey et al. [8] have proved that the k-way cut is W[1] hard parameterized by k. This hardness holds even for simple unweighted graphs. Here we use the standard parameterization by the cut size s. Getting polynomial

Ken-ichi Kawarabayashi's research is partly supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research, by C & C Foundation, by Kayamori Foundation and by Inoue Research Award for Young Scientists.

At the time of the conference, Mikkel Thorup will be Professor at the University of Copenhagen.

[1]There is a lot of confusing terminology associated with cut problems, e.g., in the original conference version of [5], "multiway cut" referred to the separation of given terminals, but that term is fortunately corrected to "multiterminal cut" in the final journal version. Here we follow the latter more explicit terminology: k-way cut for arbitrary splitting into k pieces, k-terminal cut for splitting k terminals, k-pair cut for splitting k pairs, and so forth...

parametrized by	k	s
k-way vertex cut of size s	W[1] hard [8]	W[1] hard [20]
k-way edge cut of size s	W[1] hard [8]	FPT [This paper]

Table I
FPT STATUS OF k-WAY CUT PROBLEMS

time for fixed s is trivial since we can try all subsets of s edges in $O(n^{2s})$ time. Reducing this to $O(n^s)$ time is straightforward using the sparsification from [21]. The natural challenge, raised by Marx [20], is if the k-way cut problem is FPT with parameter s. In this paper, we solve Marx's problem:

Theorem 1: The k-way cut problem is fixed-parameter tractable (FPT) when parameterized by the cut size s. More precisely, we can decide of a graph has a k-way cut with s edges in $s^{s^{O(s)}} n^2$ time, and if so, find the smallest such cut.

Corollary 2: The k-way cut problem is FPT with parameter k for any graph class with bounded average degree. In particular, this included the class of graphs excluding a specific minor.

Proof: If the average degree of a graph is bounded by d, it has a trivial k-way cut of size less than dk. ■

Cut problems and fixed-parameter tractability.: Theorem 1 offers the first FPT-separation between vertex cuts and edge cuts for a classic cut problem. When Marx [20] questioned the FPT status of k-way cut parameterized by the cut size s, he proved that the corresponding vertex cut version is W[1] hard. Our Theorem 1 states that the edge cut version in FPT. As we shall discuss below, such a difference in FPT status for edge cuts and vertex cuts is unusual.

Generally vertex cuts are at least as hard to find as edge cuts. Basically, edge cuts in $G = (V, E)$ may be found as vertex cuts in the intersection graph of $V \cup E$. Therefore, if there is a difference in FPT-status, it has to be hardness for vertex cuts and tractability for edge cuts. However, in previous work on classic cut problems, there has not been such a difference.

On the W[1] hardness side, the result of Downey et al. [8] implies that minimum k-way cut problem is W[1] hard with parameter k both for edge cuts and for vertex cuts. Table I summarizes how our new FPT results stands in contrast to the existing W[1] hardness results for k-way cut.

On the tractable side, when Marx raised our problem in [20], he also considered the k-terminal and k-pair cut problems. These are both classic NP-hard problems even for fixed $k \geq 3$ [5]. However, parameterized by the cut size s, Marx [20] proved that the k-terminal problem is FPT both for vertex and edge cuts. He also proved that the k-pair problem is FPT both for vertex and edge cuts when parameterized by both s and k. Recently this was strengthened by Marx and Razgon [19] and Bousquet et al. [3], showing that the multi-pair cut problem is FPT parameterized only by the cut size leaving the number of pairs unbounded. Also this recent FPT result holds for both vertex and edge cuts [19]. Our new result for k-way cut parameterized by the cut size breaks the established pattern, with FPT for the edge cuts contrasting Marx's W[1] hardness for the vertex cuts.

The fact that the previous FPT results hold for both vertex and edge cuts shows that our technique has to be fundamentally different. Since our problem is W[1] hard for vertex cuts, our FPT algorithm has to exploit special properties of edge cuts not shared by vertex cuts. We shall point out in Section 2.7 how our approach fails for vertex cuts. Of course, there are plenty of other reasons to expect major technical differences, e.g., the multi-pair and multi-terminal cut problems have nice linear programming relaxations dual to multi-commodity flows. In the k-way cut problem we insist on getting a certain minimal number of components rather than telling what is to be separated, and this appears harder to capture.

Below we return our focus to k-way cuts which unless otherwise stated are assumed to be edge cuts.

Planar graphs.: The k-way cut problem has been quite well-studied even in the special case of planar graphs. In the case of weighted planar graphs, Dahlhaus et al. [5] used their k-terminal cut algorithm to solve the k-way cut problem in $O(n^{3k-1} \log n)$ time. The bound was later improved to $\tilde{O}(n^{2k-1})$ time by Hartvigsen [12]. His bound was later matched by the $\tilde{O}(mn^{2k-2})$ bound by Thorup [22] for general graphs.

The case of simple unweighted planar graphs has also received attention. Hochbaum and Shmoys [14] gave an $O(n^2)$ time algorithm for simple unweighted planar graphs when $k = 3$. This was improved by He [13] to $O(n \log n)$ time. The motivation given in [14], [13] is the case of general k, with $k = 3$ being the special case for which they provide an efficient solution.

Having proved the k-way cut problem W[1] hard for simple unweighted general graphs, Downey et al. [8] asked about the FPT status for planar graphs. A positive answer follows directly from Corollary 2. However, for this case, we have a much simpler linear time algorithm:

Proposition 3: We can decide if a simple plane graph has a k-way cut with k edges in $k^{O(k)}n$ time[2], and if so, find the smallest such cut. More generally, for graph embedded with bounded genus g, we can do this in $(gk)^{O(kg^2)}n$ time.

Note that even in the special case of $k = 3$, our result improves on the above mentioned algorithms of Shmoys and Hochbaum [14] and He [13]. Proposition 3 is, however, more of an observation, where we point out that a certain planar decomposition lemma of Klein [18] provides a very nice solution to the k-way cut problem (see Section 3). Recently [6] generalized Klein's decomposition to graphs with an excluded minor. With no parallel edges, they used our observation to claim an FPT for k-way cuts with parameter k. Our Corollary 2 is much more general as it works for arbitrary graphs of bounded degrees of which simple graphs with an excluded minor is a special case.

Approximation algorithms.: The k-way cut problem has also been studied from the perspective of approximation algorithm, but no-one has been able to improve substantially on the trivial factor 2 obtained by iterative optimal 2-way cuts. The current best $2 - o(1)$ approximation [23] iterates an optimal h-way cut algorithm where $h = O(1)$, and gets an $2-h/k+O((h/k)^2)$ approximation. Getting an approximation factor strictly below 2 is a major open problem. This contrast the situation with the k-terminal cut where the approximation factor is below 1.35 [16].

Our FPT result can be seen as raising some hope that an efficient polynomial time approximation scheme (EPTAS) may be possible for the k-way cut problem. Here an EPTAS is a scheme that for any $\varepsilon > 0$, finds a $1 + \varepsilon$ approximation in $f(1/\varepsilon)n^c$ time for some function f and constant c. Generally speaking, for unweighted minimization problems, EPTAS implies FPT parameterized by the solution size [9], so our FPT is a necessary first step towards an EPTAS for k-way cut.

Techniques.: We prove Theorem 1 with a simple combinatorial algorithm not relying on any previous results. To solve the problem recursively, we will define “the powercut problem” which is much stronger than the minimum k-way cut problem. The powercut problem also generalizes the p-pair cut problem for bounded p. The p-pair problem is in itself easier in that it has long been known to be FPT for both vertex cuts and edge cuts [20]. The hard part is still to make sure that we cut into enough components. Nevertheless our power-cut problem is a nice example where a stronger inductive hypothesis gives a simpler inductive proof.

The main point in the power cut problem is that it makes it easy to handle all high degree vertices except for one, which acts like an apex vertex of the structure theorem in graph minor theory. The rest is a bounded degree graph in which we will identify a contractible edge and recurse.

2. FPT algorithm finding minimum k-way cuts of bounded size

We want to find a minimum k-way cut of size at most s. Assuming that a given graph is connected, the problem can only be feasible if $k \leq s+1$. We will use O_s to denote O assuming $s = O(1)$. We will solve the problem in $s^{s^{O(s)}}n^2 = O_s(n^2)$ time.

2.1. The powercut problem

To solve the problem k-way cut problem inductively, we are going to address a more general problem:

Definition 4: The *powercut* problem takes as input a triple (G, T, s) where G is a connected graph, $T \subseteq V(G)$ a set of terminals, and s a size bound parameter. For every $j \leq s+1$ and for every partition P of T into j sets, some of which may be empty, we want a minimal j-way cut $C_{j,P}$ of G whose sides partitions T according to P but only if there is such a feasible cut of size at most s. The resulting family $\mathcal{C}$ of minimal cuts $C_{j,P}$ is a power cut.

Powercuts generilize trivially to disconnected graphs, but it simplifies some of our arguments to require that G is conncted. Often we will identify a powercut with its set of distinct edge cuts. Note that if $|T|$ and s are bounded, then so is the total size of the power cut. More precisely,

Observation 5: The total number of edges in a power cut is bounded by $s\sum_{j=2}^{s+1}(j^{|T|}/j!) < (s+1)^{|T|+1}$.

We will show how to solve the powercut problem in quadratic time when $|T|, s = O(1)$. To solve our original problem, we k-way cut problem, we solve the powercut problem with an empty set of terminals $T = \emptyset$. In this case, for each $j \leq s+1$, we only have a single trivial partition P_j of T consisting of j empty sets, and then we return C_{k,P_k}.

We can, of course, also use our powercut algorithm to deal with cut problems related to a bounded number of terminals, e.g., the p-pair cut problem which for p pairs $\{(s_1, t_1), ..., (s_p, t_p)\}$ ask for a minimum cut that splits every pair, that is, for each i, the cut separates s_i from t_i. If there is such a cut of size at most s, we find it with a powercut setting $T = \{s_1, ..., s_p, t_1, ..., t_p\}$. In

[2]A slight improvement to $2^{O(k)}n$ time appears possible, but this is not essential to our paper which is focussed on Theorem 1.

the output powercut family $\{C_{j,P}\}$, we consider all partitions P splitting every pair, returning the smallest of the corresponding cuts. Such problems with a bounded number of terminals and a bounded cut size, but no restrictions on the number of components, are easier to solve directly, as done by Marx [20] for both edge and vertex cuts. However, for vertex cuts, Marx [20] proved that the k-way cut problem is W[1] hard. Hence the hardness is not in splitting of a bounded set of terminals, but in getting a certain number of components. With our powercut algorithm, we show that getting any specified number of components is feasible with size bounded edge cuts.

Below we will show how to solve the power cut problem recursively in quadratic time, let T_0 be the initial set of terminals, e.g., $T_0 = \emptyset$ for the k-way cut problem. We now fix

$$t = \max\{2s, |T_0|\}. \tag{1}$$

In our recursive problems we will never have more than t terminals. The parameter s will not change.

Identifying vertices and terminals.: Our basic strategy will be to look for vertices that can be identified while preserving some powercut. To make sense of such a statement, we specify a cut as a set of edges, and view each edge as having its own identity which is preserved even if its end-points are identified with other vertices. Note that when we identify vertices u and v, then we destroy any cut that would split u and v. However, the identification cannot create any new cuts. We say that u and v are *identifiable* if they are not separated by any cut of some powercut $\mathcal{C}$. It follows that if u and v are identifiable, then $\mathcal{C}$ is also a powercut after their identification, and then every powercut $\mathcal{C}'$ after the identification is also a powercut before the identification. Since loops are irrelevant for minimal cuts, identifying the end-points of an edge is the same as contracting the edge. Therefore, if the end-points of an edge are identifiable, we say the edge is *contractible*.

We do allow for the case of identifiable terminals t and t' that are not split by any cut in some powercut $\mathcal{C}$. By definition of a powercut, this must imply that there is no cut of size s separating t and t'.

Often we will identify many vertices. Generalizing the above notion, we say a set of vertex pairs is *simultaneously identifiable* if there is a powercut with no cut separating any of them. This implies that we can identify all the pairs while preserving some powercut.

Note that we can easily have cases with identifiable vertex pairs that are not simultaneously identifiable, e.g., if the graph is a path of two edges between two terminals, then either edge is contractible, yet they are not simultaneously contractible.

Recursing on subgraphs.: Often we will find identifiable vertices recursing via a subgraph $H \subseteq G$. If $\mathcal{C}$ is a powercut of G, then $\mathcal{C}|H$ denotes $\mathcal{C}$ *restricted* to H in the sense that each cut $C \in \mathcal{C}$ is replaced by its edges $C \cap E(H)$ in H, ignoring cuts that do not intersect H.

Lemma 6: Let H be a connected subgraph of G. Let S be the set of vertices in H with incident edges not in H. Define $T_H = S \cup (T \cap V(H))$ to be the terminals of H. Then each powercut $\mathcal{C}_H$ of (H, T_H, s) is the restriction to H of some powercut $\mathcal{C}$ of (G, T, s). Hence, if pairs of vertices are simultaneously identifiable in (H, T_H, s), then they are also simultaneously identifiable in (G, T, s).

Proof: Since the pairs are simultaneously identifiable in (H, T_H, s), there is a powercut $\mathcal{C}_H$ of (H, T_H, s) with no cut separating any of the pairs. Now consider a powercut $\mathcal{C}$ of (G, T, s), and let C be any cut in $\mathcal{C}$. Then $H \setminus C$ has a certain number $j \leq s+1$ of components inducing a certain partition P of T_H. In C we now replace $C \cap H$ with $C_{j,P}$ from $\mathcal{C}_H$, denoting the new cut C'. Since $C_{j,P}$ is a minimal, this can only decrease the size of C. It is also clear that $G \setminus C$ and $H \setminus C$ have the same number of components inducing the same partition of T. This way we get a powercut $\mathcal{C}'$ of (G, T, s) such that $\mathcal{C}'|H = \mathcal{C}_H$. In particular it follows that our pairs from H are simultaneously identifiable in (G, T, s). ■

2.2. Good separation

For our recursion, we are going to look for good separations as defined below. A *separation* of the graph G is defined via an edge partition into two connected subgraphs A and B, that is, each edge of G is in exactly one of A and B. We refer to A and B as the *sides* of the separation. Let S be the set of vertices in both A and B. Then S *separates* $V(A)$ from $V(B)$ in the sense that any path between them will intersect S. Contrasting vertex cut terminology, we include S in what is separated by S. In order to define a good separation, we fix

$$p = (s+1)^{t+1} \text{ and } q = 2(p+1). \tag{2}$$

Then p is the upper bound from Observation 5 on the total number of edges in a power cut with at most t terminals. The separation is *good* if $|S| \leq s$ and both A and B have at least q vertices.

Suppose we have found a good separation. Then one of A and B will contain at most half the terminals from $T \setminus S$. Suppose it is A. Recursively we will find a powercut $\mathcal{C}_A$ of A with terminal set $T_A = S \cup (T \cap V(A))$

as in Lemma 6. Finally in G we contract all edges from A that are not in the powercut $\mathcal{C}_A$.

For the validity of the recursive call, we note that $|T_A| \leq s + t/2 \leq t$. The last inequality follows because $t \geq 2s$. For the positive effect of the contraction, recall that A has at least $q = 2(p+1)$ vertices where p bounds the number of edges in $\mathcal{C}_A$. We know that A is connected, and it will remain so when we contract the edges from A that are not in $\mathcal{C}_A$. In the end, A has at most p non-contracted edges, and they can span at most $p+1$ distinct vertices. The contractions thus reduce the number of vertices in G by at least $|A| - p - 1$.

Below, splitting into a few cases, we will look for good separations to recurse over.

2.3. Multiple high degree vertices

A vertex is said to have *high degree* if it has at least

$$d = q + s - 1 \qquad (3)$$

neighbors. Here q was the lower bound from (2) on the number of vertices in a side of a good separation. Suppose that the current graph G has two high degree vertices u and v. In that case, we check if there is a minimal cut D between u and v of size at most s. If not, we can trivially identify u and v and recurse.

If there is a minimal cut D of size at most s between u and v, let A be the component containing u in $G \setminus D$, and let B be the subgraph with all edges not in A.

Lemma 7: The subgraphs A and B form a good separation.

Proof: The set S of vertices in both A and B are exactly the end-points on the u-side of the edges in D. Therefore $|S| \leq |D| \leq s$. We now need to show that each of A and B span at least q vertices. This is trivial for B since B contains v plus all the $d = q + s - 1$ neighbors of v. For the case of A, we note that D can separate u from at most s of its neighbors. This means that u is connected to at least $d - s = q - 1$ vertices in $G \setminus D$, so A contains at least q nodes. ■

Thus, if we have two high degree vertices, depending on the edge connectivity between u and v, we can either just identify u and v, or recurse via a good separation. Below we may therefore assume that the graph has at most one high degree vertex.

2.4. No high degree vertex

We now assume that there is no vertex with high degree $\geq d$—c.f. (3). The case of one high degree "apex" vertex will later be added a straightforward extension.

A kernel with surrounding layers.: We start by picking a start vertex v_0 and grow an arbitrary connected subgraph H_0, called the *kernel* from v_0 such that H_0 contains all edges leaving v_0 and H_0 spans $h \geq d$ vertices where $h = O_s(1)$ is a parameter to be fixed later. Next we pick edge disjoint minimal *layers* H_i, $i = 1, ..., p$, subject to the following constraints:

(i) The layer H_i contains no edges from $H_{<i} = \bigcup_{j<i} H_j$, but H_i contains all other edges from G incident to the vertices in $H_{<i}$.

(ii) Each component of H_i is either a *big component* with at least q vertices, or a *limited component* with no edge from $G \setminus H_{\leq i}$ leaving it—if a component is both, we view it as big.

We note that the layer H_i is typically not unique as there may be many ways of chosing the minimally big components. Recall that a powercut $\mathcal{C}$ can have at most p edges. This means that there must be at least one of the $p+1$ edge disjoint subgraphs H_i which has no edges in $\mathcal{C}$. Supposing we have guessed this H_i, we will find a set F_i of simultaneously contractible edges from H_0 (note that we mean H_0, not H_i). By definition, $F_0 = E(H_0)$. If $i > 0$, condition (i) implies that H_i is a cut between $H_{<i}$ and the rest of the graph, and we will use this fact to find the set F_i. Since one of the guesses must be correct, the intersection $F = \bigcap_{i=0}^{p} F_i = \bigcap_{i=1}^{p} F_i$ must be simultaneously contractible. An alternative outcome will be that we find a good separation which requires q vertices on either side. This is where condition (ii) comes in, saying that we have to grow each component of H_i until either it becomes a big component with q vertices, or it cannot be grown that big because no more edges are leaving it.

Before elaborating on the above strategy, we note that the graphs H_i are of limited size:

Lemma 8: The graph $H_{\leq i}$ has at most hd^i vertices.

Proof: We prove the lemma by induction on i. By definition $|V(H_0)| = h$. For the inductive step with $i > 0$, we prove the more precise statement that layer H_i has at most d times more vertices than the vertices it contains from layer H_{i-1}.

Since each layer is a cut, the edges leaving $H_{<i}$ must all be incident to H_{i-1}. We will argue that each component of H_i has at most d vertices. This is trivially satisfied when we start, since each vertex from H_{i-1} comes with its at most $d-1$ neighbors. Now, if we grow a component along an edge, it is because it has less than $q < d$ vertices, including at least one from H_{i-1}. Either the edge brings us to a new vertex, increasing the size of the component by 1, which is fine, or the edge connects to some other component from H_i which by induction also had at most d vertices per vertex in H_{i-1}. ■

If $V(G) = V(H_{\leq p})$, then G has only $hd^i = O_s(1)$ vertices, and then we can solve the powercut problem exhaustively. Below we assume this is not the case.

Pruning layers checking for good separations.: Consider a layer H_i, $i > 0$, and let H_i^- be the union of the big components of H_i. Moreover let $H_{<i}^+$ be $H_{<i}$ combined with all the limited components from H_i. We call H_i^- the pruned layer. Since the limited components have no incident edges from $G \backslash H_{\leq i}$, we note that H_i^- is a cut between $H_{<i}^+$ and the rest of the graph. The lemma below summarizes the important properties obtained:

Lemma 9: For $i = 1, ..., p$:

(i) Pruned layer $H_i^- \subseteq H_i$ is a cut separating H_0 from $G \setminus V(H_{\leq i})$. In particular, we get an articulation point if we identify all of H_i^- in a single vertex.
(ii) Each component of H_i^- is of size at least q.

Now, we take each pruned layer H_i^- separately, and order the components arbitrarily. For every pair A and B of consecutive components (of order at least q), we check if the edge connectivity between A and B is at least s in G. If not, there is a minimal cut D in G with at most s edges which separates A and B. We claim this leads to a good separation. On the one side of the separation, we have the component $\overline{A}$ of $G \setminus D$ containing A, and on the other we have the reminder $\overline{B}$ of G which includes B and cut edges from D. Then $\overline{A}$ and $\overline{B}$ intersect in at most $|D| \leq s$ vertices, and both $\overline{A}$ and $\overline{B}$ have at least q vertices. Thus we get a good separation.

Below we assume for each pruned layer that the edge connectivity between consecutive components is at least s.

Articulation points from pruned layers.:

Lemma 10: If there is a powercut of (G, T, s) that does not use any edge from H_i, then all vertices in the pruned layer H_i^- can be identified in a single vertex v_i.

Proof: We are claiming that no cut D from $\mathcal{C}$ separates any vertices from H_i^-. Otherwise, since D does not contain any edges from H_i, the cut would have to go between components from H_i^-. In particular, there would be two consecutive components of H_i^- separated by D. However, D has at most s edges, and we already checked that there was no such small cut between any consecutive components of H_i^-. ■

Below we assume we have guessed a layers H_i that is not used in some powercut of (G, T, s). Let $[H_i^- \mapsto v_i]$ denote that all vertices from H_i^- are identified in a single vertex v_i, which we call the *articulation point.* From Lemma 10 it follows that some powercut is preserved in $(G[H_i^- \mapsto v_i], T[H_i^- \mapsto v_i], s)$.

Next, from Lemma 9 (i) we get that $H_{\leq i}[H_i^- \mapsto v_i]$ is a block of $G[H_i^- \mapsto v_i]$ separated from the rest by the articulation point v_i. As in Lemma 6, we now find a powercut $\mathcal{C}_i$ of $\big(H_{\leq i}[H_i^- \mapsto v_i],\ \{v_0\} \cup (T \cap V(H_{\leq i}))[H_i^- \mapsto v_i],\ s\big)$. The powercut $\mathcal{C}_i$ can be found exhaustively since $H_{\leq i}$ has at most $hd^i = O_s(1)$ vertices. Since this is not a recursive call, so it is OK if $\{v_0\} \cup (T \cap V(H_{\leq i}))[H_i^- \mapsto v_i]$ involves $t + 1$ terminals.

Lemma 11: If there is a powercut of (G, T, s) that does not use any edge from H_i, then there is such a powercut which agrees with $\mathcal{C}_i$ on $H_{\leq i}$, and on H_0 in particular.

Proof: From Lemma 6 we get that $\mathcal{C}_i$ is the restriction to $H_{\leq i}[H_i^- \mapsto v_i]$ of some powercut $\mathcal{C}_i'$ of $(G[H_i^- \mapsto v_i], T[H_i^- \mapsto v_i], s)$. From Lemma 10 it follows that $\mathcal{C}_i'$ is also a powercut of (G, T, s). Since $\mathcal{C}_i'$ does not contain any edges contracted in H_i^-, we conclude that $\mathcal{C}_i$ is the restriction of $\mathcal{C}_i'$ to $H_{\leq i}$. ■

With our assumption that H_i is not used in some powercut, we get that all edges in $F_i = E(H_0) \setminus E(\mathcal{C}_i)$ are identifiable. Disregarding the assumption, we are now ready to prove

Lemma 12: Let $F = \bigcap_{i=1}^{p} F_i$ be the set of edges from H_0 that are not used in any $\mathcal{C}_i$, $i = 1, ..., p$. The edges from F are simultaneously contractible.

Proof: Given any powercut $\mathcal{C}$ of (G, T, s), since it has at most p edges, we know there is come $i \in \{0, ..., p\}$ such that $\mathcal{C}$ does not use any edge from H_i. If i is 0, this means all edges from H_0 are contractible. For any other i, the claim follows from Lemma 11. ■

With Lemma 12 we contract all edges from H_0 that are not in some $\mathcal{C}_i$. From Observation 5, we know that each $\mathcal{C}_i$ involves at most $(s+1)^{t+2}$ edges, so combined they involve at most $p(s+1)^{t+2}$ un-contracted edges, spanning at most $p(s+1)^{t+2} + 1$ distinct vertices. As the initial size for H_0, we start with

$$h = 2(p(s+1)^{t+2} + 1) \tag{4}$$

vertices. Therefore, when we contract all edges from H_0 that are not in some $\mathcal{C}_i$, we get rid of half the vertices from H_0.

2.5. A single high degree "apex" vertex

All that remains is to consider the case where there is a single high degree vertex r with degree $\geq d$—c.f. (3). We are basically going to run the reduction for no high degree from Section 2.4 on the graph $G \setminus \{r\}$, but with some subtle extensions described below.

Starting from an arbitrary vertex v_0 that is neighbor to r, we construct the layers H_i in $G \setminus \{r\}$. If this includes all vertices of $G \setminus \{r\}$, then G has $O_s(1)$ vertices, and then we find the powercut exhaustively.

Next we add the vertex r to each layer H_i, including all edges between r and $H_i \setminus H_{<i}$. We denote this graph H_i^r. Note that H_0^r is connected since r is a neighbor of v_0. Also note that all the H_i^r are edge disjoint like the H_i.

After the addition of r, for $i > 0$, we consider any limited component involving r big. More precisely, in H_i^r we say that a component is *big* if it is has q or more vertices or if it contains r. The remaining components are *limited*. Removing all remaining limited components from H_i^r we get the *pruned layer* H_i^{r-}. Similarly, we have the graph $H_{<i}^{r+}$ which is $H_{<i}^r$ expanded with the limited components from H_i^r. Corresponding to Lemma 9, we get

Lemma 13: For $i = 1, ..., p$:

(i) The vertices from the pruned layer H_i^{r-} form a vertex separator in G between H_0^r and $G \setminus H_{\leq i}^r$. In particular, we get an articulation point if we identify H_i^{r-} in a single vertex.

(ii) Each component of H_i^{r-} which does not contain r has at least q vertices.

Proof: Above (ii) is trivial. Concerning (i), we already have from Lemma 9 that the edges from H_i^- provide a cut of $G \setminus \{r\}$ between H_0 and $G \setminus \{r\} \setminus H_{\leq i}$. The vertices in H_i^- provide a corresponding vertex separation in $G \setminus \{r\}$. When adding r to the graph and to the separation, we get a vertex separation $V(H_i^{r-})$ between H_0^r and $G \setminus H_{\leq i}^r$. ■

Now, as in Section 2.4, we order the components of H_i^{r-} arbitrarily, and check if the edge connectivity between pairs of consecutive components is at least s in G. If not, we claim there is a good separation. Let D be a cut in G of size at most s between two components A and B of H_i^{r-}. If A and B both have at least q vertices, then we have the same good separation as in Section 2.4. Otherwise, one of them, say B involves r. In this case we have an argument similar to that used for two high degree vertices in Section 2.3. On one side of the good separation, we have the component $\overline{A}$ of $G \setminus D$ including A. Clearly it has at least $|V(A)| \geq q$ vertices. The other side $\overline{B}$ is the rest of G including B and the cut edges from D. Then $\overline{B}$ includes the neighborhood of all vertices in B including all neighbors of r, so $\overline{B}$ has at least $d + 1 > q$ vertices. Below we assume that we did not find such a good separation.

We now continue exactly as in Section 2.4. For $i = 1, ..., q$ we identify the vertices of H_i^{r-} in a vertex v_i which becomes an articulation point, and then we find a powercut $\mathcal{C}_i$ of $\big((H_{\leq i}^r[H_i^{r-} \mapsto v_i], \{v_0\} \cup (T \cap V(H_{\leq i}^r))[H_i^{r-} \mapsto v_i]), s\big)$. Corresponding to Lemma 12, we get

Lemma 14: The edges from H_0^r that are not used in any $\mathcal{C}_i$, $i = 1, ..., p$ are simultaneously contractible.

As in Section 2.4, we conclude that we get at most $p(s+1)^{t+2}$ un-contracted edges from Lemma 14 and they span at most $p(s+1)^{t+2} + 1$ distinct vertices. As the initial size for H_0^r, we start with $h + 1 = 2(p(s+1)^{t+2} + 1)$ vertices. Then the contractions of Lemma 14 allows us to get rid of at least half the h vertices in H_0^r.

2.6. *Analysis and implementation*

We are now going to analyze the running time including some implementation details of the above recursive algorithm, proving a time bound of

$$T(n) = O\left(s^{t^{O(t)}} n^2\right) = s^{t^{O(t)}} n^2. \qquad (5)$$

First, we argue that we can assume sparsity with at most $O(sn)$ edges. More precisely, if the graph at some point has $m \geq 2sn$ edges, then, as in [21], we find s edge disjoint maximal spanning forests. If an edge (v, w) is not in one of these spanning forests, then v and w are s edge connected. We can therefore contract all such outside edges, leaving us with at most $sn \leq m/2$ edges. This may also reduce the number of vertices, which is only positive. Even if this sparsification process is applied to several subproblems in our recursion, the overall sparsification cost is $O(sn^2)$.

In our analysis, for simplicity, we just focus on the case with no high degree vertices from Section 2.4. When we look for good separations, we check if the edge connectivity between two vertex sets is s. As we saw above, the graph can be assumed to have at most $2ns$ edges, so this takes only $O(s^2 n)$ time [10] including identifying a cut with s edges if it exists. The number of such good separation checks is limited by the total number of components in all the layers H_i, and for each layer, this is limited by the number of vertices. Thus, by Lemma 8, we have at most $\sum_{i=1}^{p} hd^i < 2hd^p$ good separation checks, each of which takes $O(s^2 n)$ time. With $t \geq 2s$, $p = (s+1)^{t+1}$, $q = 2(p+1)$, $d = q + s - 1$, and $h = 2(p(s+1)^{t+2} + 1)$—c.f. (1), (2), (3), and (4)—we get that the total time for good separation checks is bounded by

$$O(2hd^p s^2 n) = O(s^{t^{O(t)}} n).$$

If we do find a good separation, we recurse on one of the sides A, which we know has at least q vertices. Including the separating vertices, we know that A has at most $n - q + s$ vertices. After the recursion, we can identify all but $q/2$ vertices in A. All this leads to a the recurrence

$$T(n) \leq \max_{q\leq \ell \leq n-q+s} O\left(s^{t^{O(t)}} n\right) + T(\ell) + T(n-\ell+q/2).$$

Inductively this recurrence satisfies (5), the worst-case being when ℓ attains one of its extreme values.

If we do not find a good separation, for $i = 1, ..., p$, we exhaustively find a powercut of a graph with at most hd^i vertices and shd^i edges. We simply consider all the $(shd^i)^s$ potential cuts with s edges, and that is done in $O\left(s^{t^{O(t)}} n\right)$ total time. This reduces the number of vertices in H_0 from h to $h/2$. In particular, we get rid of at least one vertex, so this is also a recurrence satisfying (5). This completes the proof that (5) bounds our overall running time. In the case of the k-way cut problem, we start with no terminals. Then $t = 2s$, and then our running time is bounded by $O\left(s^{s^{O(s)}} n^2\right)$. This completes the proof of Theorem 1.

2.7. *Failure for vertex cuts*

Recall that contrasting the FPT algorithms for cutting pairs and terminals [3], [20], [19], our algorithm for k-way cuts has to be specific to edge cuts since the vertex cut version is W[1] hard [20]. Our power cuts themselves could be defined equally well for vertex cuts, but in the vertex cut case, they offer no easy way of handling high degree vertices. To be more precise, the idea in a good separation was that each side was so large that a power cut would leave some edges for later contraction. A high degree vertex was itself enough for a large side when dealing with bounded size edge cuts. However, a bounded size vertex cut could take out the high degree vertex with all its incident edges.

With bounded degree vertices, we could potentially perform a recursive step from Section 2.4 even with vertex cuts. However, the point in the recursive step is to identify vertices, thus creating higher degree vertices for later recursive steps.

3. Planar graphs and bounded genus graphs

For k-way cut in simple planar graphs Corollary 2 gives a quadratic algorithm when k is fixed. However, it turns out that there is a much simpler but overlooked linear time algorithm. We observe here that a k-way cut algorithm follows nicely from a certain decomposition lemma that Klein [18] developed for his approximate TSP algorithm. In fact, this seems to be the simplest direct application of Klein's lemma for a classic problem. The k-way cut algorithm follows whenever we have Klein's decomposition (which is dual to Baker's layered approach [1] to planar graphs). We present a faster such decomposition for bounded genus graphs. Recently Demaine et al. [6] generalized Klein's decomposition to graph classes with an excluded minor. They cite our observation (Proposition 18 below) for the application to k-way cuts. Our Corollary 2 is, of course, much more general as it does not need an excluded minor.

We now present the simple algorithm for the planar case using several known ingredients.

Observation 15: A simple planar graph has a k-way cut of size at most $5(k - 1)$.

The same observation was used in the previous slower algorithms for 3-way cuts [13], [14]. Our k-way cut algorithm applies to planar graphs with parallel edges, but like our algorithm for general graphs, it needs a bound s on the size of the cuts considered. From Observation 15, we get $s = 5k - 5$. Internally, the algorithm will consider minors which may have many parallel edges and no small cuts, but we will only look for cuts of size at most s.

Our algorithm uses the notion of tree decompositions and tree-width. The formal definitions are reviewed in Appendix, which also includes the proof of the lemma below which is kind of folklore:

Lemma 16: If a graph H has tree width at most w, then we can find a minimum k-way cut in $k^2 w^{O(w)}|V(H)|)$ time.

For any set A of edges, we let G/A denote G with the edges A contracted. If G is embedded, respecting the embedding, we contract the edges from A one by one, except that loops are deleted. We need the following theorem:

Lemma 17 (Klein [18]): For any parameter q and a planar graph G with n vertices, there is an $O(n)$ time algorithm to partition the edges of G into q disjoint edge sets S_0,...,S_{q-1} such that for each $i \in [q]$, the graph G/S_i has tree width $O(q)$.

We now observe the simple application to k-way cuts.

Proposition 18: Given a graph G, suppose we have a partition the edges of G into q disjoint edge sets S_0,...,S_{q-1} such that for each $i \in [q]$, the graph G/S_i has tree width $O(q)$. We can then find a minimum k-way cut of size up to $q - 1$ in $k^2 q^{O(q)} n$ time.

Proof: The algorithm is trivial. We use Lemma 16 with $w = O(q)$ to compute the minimum k-way cut D_i of each G/S_i and return the smallest of these cuts D_i.

Cutting after some edges have been contracted is also a cut in the original graph, so the cut returned by our algorithm is indeed a k-way cut. We need to argue that one of the D_i is a minimal one for G. However, we are only interested if the minimum cut is of size at most s, which means that it must be disjoint from at least one of the $q \geq s+1$ disjoint S_i. Then D is also a k-way cut of G/S_i. Hence the minimum k-way cut D_i of G/S_i is also a minimum k-way cut of G. ■

From Observation 15, Lemma 17, and Proposition 18 with $q = 5(k-1)+1$, it follows that we can solve the k-way cut problem for a simple plane graph in $O(k^{O(k)}n)$ time. This is the planar part of Proposition 3.

Bounded genus.: We now extend our planar algorithm to a graph embedded with bounded genus. From Euler's formula, we get

Observation 19: A simple graph embedded into a surface with genus g and $n = |V(G)| \geq 6g + k$ has a minimum k-way cut in G of size at most $6k - 6$.

Next we need the following generalization of Klein's Lemma 17:

Lemma 20: For any parameter q and a graph G embedded into a surface of genus g with n vertices, there is an $2^{O(g^2 q)}n$ time algorithm to partition the edges of G into q disjoint edge sets S_0,...,S_{q-1} such that for each $i \in [q]$, the graph G/S_i has tree width $O(g^2 q)$.

Lemma 20 with a partition time of $O(g^3 n \log n)$ follows from [4], [7]. Our time bound is better when $g, q = O(1)$. Our proof of Lemma 20 is deferred to the full version of this paper. Applying Proposition 18 as in the planar case, we solve the k-way cut problem for a simple unweighted graph embedded with genus g in $(gk)^{O(kg^2)}n$ time. Thus we get the bounded genus part of Proposition 3.

In fact, we can also plug Lemma 20 back into Klein's original approximate TSP algorithm, generalizing his linear time solution from the planar to the bounded genus case.

References

[1] B. Baker, "Approximation algorithms for NP-complete problems on planar graphs," *J. ACM*, vol. 41, no. 1, pp. 153–180, 1994.

[2] H. L. Bodlaender, "A linear-time algorithm for finding tree-decompositions of small treewidth," *SIAM J. Comput.*, vol. 25, no. 6, pp. 1305–1317, 1996.

[3] N. Bousquet, J. Daligault, and S. Thomassé, "Multicut is FPT," in *STOC'11*, 2011, pp. 459–468. Also arXiv:1010.5197v1 [cs.DS].

[4] S. Cabello and E. Chambers, "Multiple source shortest paths in a genus g graph," in *Proc. 18th SODA*, 2007, pp. 89–97.

[5] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis, "The complexity of multiterminal cuts," *SIAM J. Comput.*, vol. 23, no. 4, pp. 864–894, 1994.

[6] E. Demaine, M. Hajiaghayi, and K. Kawarabayashi, "Contraction decomposition in h-minor-free graphs and algorithmic applications," in *STOC'11*, 2011, pp. 441–450.

[7] E. Demaine, M. Hajiaghayi, and B. Mohar, "Approximation algorithms via contraction decomposition," in *Proc. 18th SODA*, 2007, pp. 278–287.

[8] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto, and F. Rosamond, "Cutting up is hard to do: the parameterized complexity of k-cut and related problems," *Electr. Notes Theor. Comput. Sci.*, vol. 78, 2003.

[9] R. Downey and M. Fellows, *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999.

[10] L. Ford and D. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[11] O. Goldschmidt and D. S. Hochbaum, "A polynomial algorithm for the k-cut problem for fixed k," *Math. Oper. Res.*, vol. 19, no. 1, pp. 24–37, 1994, announced at FOCS'88.

[12] D. Hartvigsen, "Minimum path basis," *J. Algorithms*, vol. 15, no. 1, pp. 125–142, 1993.

[13] X. He, "An improved algorithm for the planar 3-cut problem," *J. Algorithms*, vol. 12, no. 1, pp. 23–37, 1991.

[14] D. S. Hochbaum and D. Shmoys, "An $o(|v|^2)$ algorithm for the planar 3-cut problem," *SIAM J. Algebraic and Discrete Methods*, vol. 6, pp. 707–712, 1985.

[15] Y. Kamidoi, N. Yoshida, and H. Nagamochi, "A deterministic algorithm for finding all minimum k-way cuts," *SIAM J. Computing*, vol. 36, no. 5, pp. 1329–1341, 2006.

[16] D. Karger, P. Klein, C. Stein, , M. Thorup, and N. Young, "Rounding algorithms for a geometric embedding of minimum multiway cut," *Math. Oper. Res.*, vol. 29, no. 3, pp. 436–461, 2004.

[17] D. R. Karger and C. Stein, "A new approach to the minimum cut problem," *J. ACM*, vol. 43, no. 4, 1996.

[18] P. Klein, "A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights," *SIAM J. Comput.*, vol. 37, no. 6, pp. 1926–1952, 2008.

[19] D. Marx and I. Razgon, "Fixed-parameter tractability of multicut parameterized by the size of the cutset," in *STOC'11*, 2011, pp. 469–478. Also arXiv:1010.3633v1 [cs.DS].

[20] D. Marx, "Parameterized graph separation problems," *Theor. Comput. Sci.*, vol. 351, no. 3, pp. 394–406, 2006, announced at IWPEC'04.

[21] H. Nagamochi and T. Ibaraki, "Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph," *Algorithmica*, vol. 7, pp. 583–596, 1992.

[22] M. Thorup, "Minimum k-way cuts via deterministic greedy tree packing," in *Proc. 40th STOC*, 2008, pp. 159–166.

[23] M. Xiao, L. Cai, and A. C. Yao, "Tight approximation ratio of a general greedy splitting algorithm for the minimum *k*-way cut problem," *Algorithmica*, vol. 59, no. 4, pp. 510–520, 2011.

APPENDIX

In this appendix, we shall deal with the tree width bounded case. There is nothing technically new. We are just specializing standard techniques for bounded tree-width to k-way cuts.

Recall that a *tree decomposition* of a graph G is a pair (T, R), where T is a tree and R is a family $\{R_t \mid t \in V(T)\}$ of vertex sets $R_t \subseteq V(G)$, such that the following two properties hold:

(1) $\bigcup_{t\in V(T)} R_t = V(G)$, and every edge of G has both ends in some R_t.

(2) If $t, t', t'' \in V(T)$ and t' lies on the path in T between t and t'', then $R_t \cap R_{t''} \subseteq R_{t'}$.

As noted in [2] we can assume that T is a rooted binary tree with $O(n)$ nodes.

The *width* of a tree-decomposition is $\max |R_t|$ for $t \in V(T)$. The *tree width* of G is defined as the minimum width taken over all tree decompositions of G. We often refer to the sets R_t as a *bags* of the tree decomposition.

We first observe that if a given graph has tree-width at most w, then we can construct a tree-decomposition of width at most w in $O(w^w n)$ time by Theorem 21 below.

Theorem 21 ([2]): For any constant w, there exists an $O(w^w n)$ time algorithm that, given a graph G, either finds a tree-decomposition of G of width w or concludes that G has tree width at least w. For a plane graph, the time complexity can be improved to $O(2^w n)$.

To complete the proof of Lemma 16, we just need to prove the following:

Lemma 22: Given a tree-decomposition (T, R) of G of width at most w, then for any k, we can find a minimum k-way cut in $k^2 w^{O(w)}|V(G)|$ time.

Proof: For a node $t \in T$, we let G_t be the subgraph of G induced by the bags of the subtree descending from t. We use a dynamic program that for each node $t \in T$ computes the following information: for every partition P of R_t and every $r < k$, we compute the a minimal $(r + |P|)$-way cut D of G_t which respects the partition P in the sense that for each $A \in P$, there is a component H of $G_t \setminus D$ with $H \cap R_t = A$. In addition we have r components not intersecting R_t. Since $|R_t| \le w$, the number of such (P, r) is bounded by $w^w k$. The root node t of T will contain a minimum k-way cut of $G_t = G$ among the combinations (P, r) with $|P| + r = k$.

We assume we have this information for each child t' and t'' of t. A simple brute force solution goes as follows. We try combining each solution (P', r') for t', and (P'', r'') for t'' with each partition Q of R_t induced by subgraphs of $G|R_t$. We can easily check if these three objects are consistent, and if so, they represent a solution for a (P, r) for t. For each (P, r) we keep the smallest such solution. The number of combinations to consider is $(kw^w)^2 w^w$, and each can be checked in $O(w^2)$ time, so the total time needed to compute the information for t is $k^2 w^{O(w)}$. ■

It appears fairly easy to improve the bound in Lemma 22 to $k^2 2^{O(w)}|V(G)|$ when the bounded tree-width graph G is planar. The point is to only consider partitions over w vertices that are on the outer face of a plane embedding, for these can only be partitioned in $2^{O(w)}$ ways. This is essentially done by [18] but in a way that appears a bit specialized for TSP. We found several works on such geometric tree decompositions, but the constructions address larger w and are not linear in n. Anyhow, this is only a secondary issue for this paper where $w = O(1)$. The far more important part is the results for general graphs in Theorem 1.