Solving Multicut Faster than 2^n

Daniel Lokshtanov[1], Saket Saurabh[1,2], and Ondřej Suchý[3]

[1] University of Bergen, Norway. `daniello@ii.uib.no`
[2] Institute of Mathematical Sciences, India. `saket@imsc.res.in`
[3] Czech Technical University in Prague, Czech Republic. `ondrej.suchy@fit.cvut.cz`

Abstract. In the Multicut problem, we are given an undirected graph $G = (V, E)$ and a family $\mathcal{T} = \{(s_i, t_i) \mid s_i, t_i \in V\}$ of pairs of requests and the objective is to find a minimum sized set $S \subseteq V$ such that every connected component of $G \setminus S$ contains at most one of s_i and t_i for any pair $(s_i, t_i) \in \mathcal{T}$. In this paper we give the first non-trivial algorithm for Multicut running in time $\mathcal{O}(1.987^n)$.

1 Introduction

Cuts and flows represent one of the most fundamental fields of studies in network design. Given two distinguished vertices in a graph one can determine the minimum size of a vertex or edge cut separating them in polynomial time using the well known min-cut max-flow duality. However, when one wants to separate more than two terminals from each other, the duality no longer works and the problem of determining the smallest size cut becomes NP-hard for every fixed number of at least three terminals [8].

In this paper we consider a generalization of the above problem, where one is given several pairs of vertices (*requests*) and the task is to determine the minimum size of a set of vertices that separates each pair. More formally, we consider the following problem:

Multicut
Input: An undirected graph $G = (V, E)$ and a family $\mathcal{T} = \{(s_i, t_i) \mid s_i, t_i \in V\}$.
Task: Find a minimum size set $S \subseteq V$ such that every connected component of $G \setminus S$ contains at most one of s_i and t_i for any pair $(s_i, t_i) \in \mathcal{T}$.

Note specifically, that we allow the terminals itself to be deleted, i.e., we consider the unrestricted variant of the problem as named in [2]. However, the version where the terminals are forbidden to delete reduces to the version we investigate (see Observation 1 for more details). If a set $S \subseteq V$ has the requested properties, than we call it a *cut-set* for $(G, \mathcal{T})$. As the roles of s_i and t_i in the pairs of $\mathcal{T}$ are symmetric, we consider the pairs unordered, i.e., if $(s, t) \in \mathcal{T}$, then we also say $(t, s) \in \mathcal{T}$. Moreover, for technical reasons we allow $s = t = v$ for a pair $(s, t) \in \mathcal{T}$ and $v \in V$. In this case, obviously, v must be in any cut-set, as otherwise the component containing v contains both s and t.

MULTICUT generalizes MULTIWAY CUT, where one is given a set of terminals and the task is to separate each two of them. Therefore, MULTICUT is NP-hard already for three requests [8]. Moreover, there is no constant factor approximation for the problem, unless the Unique Games Conjecture fails [3]. Furthermore, the edge variant of MULTICUT is MaxSNP-hard already on stars [12].

Hence, we turn our attention to exact algorithms working in exponential time. Since one can try all subset of vertices, the problem admits a trivial $\mathcal{O}(2^n n^3)$-time algorithm, where n is the number of vertices of the input graph. In this work we break the 2^n barrier. Namely we prove the following theorem.

Theorem 1. MULTICUT *on an n vertex graph can be solved in $\mathcal{O}(1.987^n)$-time.*

Related work. Deleting vertices of the input graph such that the resulting graph satisfies some interesting properties is one of the most well studied directions in exact exponential algorithms. This includes the classical $\mathcal{O}(1.2109^n)$-time algorithm of Robson [18] for MAXIMUM INDEPENDENT SET, an $\mathcal{O}(1.7356^n)$-time algorithm for FEEDBACK VERTEX SET [19], or an $\mathcal{O}(1.4689^n)$-time algorithm for DOMINATING SET [13], to name at least a few of them. For MULTIWAY CUT, such an algorithm was presented by Fomin et al. [11] achieving $\mathcal{O}(1.8638^n)$-time. This was recently improved to $\mathcal{O}(1.4766^n)$-time by Chitnis et al. [5]. For MULTICUT, however, no algorithm faster than the trivial $\mathcal{O}(2^n \cdot n^{\mathcal{O}(1)})$-time is known.

Concerning approximation algorithms, Garg et al. [12] show that MULTICUT can be approximated within $\mathcal{O}(\log k)$ factor, where $k = |\mathcal{T}|$. Cut problems are also well studied from the perspective of parameterized algorithms. Marx [14] was the first to consider cut problems in the context of parameterized complexity. He gave an algorithm for parameterized MULTIWAY CUT with running time $\mathcal{O}(4^{k^3} n^{\mathcal{O}(1)})$ with the current fastest algorithm running in time $\mathcal{O}(2^k n^{\mathcal{O}(1)})$ [7]. The notions used in this paper has been useful in settling parameterized complexity of variety of problems including DIRECTED FEEDBACK VERTEX SET [4], ALMOST 2 SAT [17] and ABOVE GUARANTEE VERTEX COVER [17,16]. Recently, Marx and Razgon [15] and Bousquet, Daligault and Thomassé [1] independently showed that MULTICUT, finding k vertices to disconnect given pairs of terminals is FPT. Continuing this line of study, Chitnis, Hajiaghayi and Marx studied MULTIWAY CUT on directed graphs and showed it to be FPT [6].

2 Preliminaries

Our notation for graph theoretic notions is standard. We summarize some of the frequently used concepts here. For a finite set V, a pair $G = (V, E)$ such that $E \subseteq V^2$ is a graph on V. The elements of V are called *vertices*, while pairs of vertices $\{u, v\}$ such that $\{u, v\} \in E$ are called *edges*. In the following, let $G = (V, E)$ and $G' = (V', E')$ be graphs, and $U \subseteq V$ some subset of vertices of G. Let G' be a subgraph of G. If E' contains all the edges $\{u, v\} \in E$ with $u, v \in V'$, then G' is an *induced subgraph* of G, *induced by* V', denoted by $G[V']$.

For any $U \subseteq V$, $G \setminus U = G[V \setminus U]$. For $v \in V$, $N_G(v) = \{u \mid \{u,v\} \in E\}$. A set of vertices C of V is said to be *clique* if there is an edge for every pair of vertices in C.

3 Basic observations

Our algorithm applies the following two operations on vertices of the graph:

Definition 1. *1. By* deleting *a vertex v from the instance $(G, \mathcal{T})$ we mean removing the vertex v from the vertex set of G together with all its incident edges as well as removing all pairs containing v from $\mathcal{T}$. I.e., we continue with the instance $(G', \mathcal{T}')$, where $G' = G \setminus \{v\}$ and $\mathcal{T}' = \mathcal{T} \setminus \{(u,v) \mid u \in V\}$.*

2. By contracting *a vertex v in the instance $(G, \mathcal{T})$ we mean first turning the neighborhood of v into a clique and adding into $\mathcal{T}$ the pair (u,w), whenever (v,w) was in T and u is a neighbor of v in G. Finally, we remove the vertex v from V together with all its incident edges from E and all pairs containing v from $\mathcal{T}$. I.e., we continue with the instance $(G', \mathcal{T}')$, where $G' = (V \setminus \{v\}, E')$, $E' = E \cup \{\{u,w\} \mid u,w \in N_G(v)\} \setminus \{\{v,u\} \mid u \in V\}$, and $\mathcal{T}' = \mathcal{T} \cup \{(u,w) \mid u \in N_G(v) \wedge (v,w) \in \mathcal{T}\} \setminus \{(v,w) \mid w \in V\}$.*

The following two lemmata show, that the two operations correspond to taking and not taking the vertex into the constructed solution, respectively.

Lemma 1. *If $(G', \mathcal{T}')$ is obtained from $(G, \mathcal{T})$ by deleting a vertex v, then $S \subseteq V$ containing v is a cut-set for $(G, \mathcal{T})$ if and only if $S \setminus \{v\}$ is a cut-set for $(G', \mathcal{T}')$.*

Proof. First, if S is a cut-set for $(G, \mathcal{T})$, then $S \setminus \{v\}$ is a cut-set for $(G', \mathcal{T}')$, since we have $G' \setminus (S \setminus \{v\}) = G \setminus S$ and $\mathcal{T}' \subseteq \mathcal{T}$.

Let us now assume, that $S \setminus \{v\}$ is a cut-set for $(G', \mathcal{T}')$ and $v \in S$. Then again $G \setminus S = G' \setminus (S \setminus \{v\})$, each connected component of $G \setminus S$ contains at most one vertex of each pair in $\mathcal{T}'$ and, since all pairs in $\mathcal{T} \setminus \mathcal{T}'$ contain v, also of each pair in $\mathcal{T}$. $\square$

Lemma 2. *If $(G', \mathcal{T}')$ is obtained from $(G, \mathcal{T})$ by contracting a vertex v and $(v,v) \notin \mathcal{T}$, then $S \subseteq V \setminus \{v\}$ is a cut-set for $(G, \mathcal{T})$ if and only if S is a cut-set for $(G', \mathcal{T}')$.*

Proof. Suppose first that $S \subseteq V \setminus \{v\}$ is not a cut-set for $(G, \mathcal{T})$. Hence there is a pair $(x,y) \in \mathcal{T}$ such that there is an x-y-path in $G \setminus S$. If this path does not contain v then it is also present in $G' \setminus S$, $(x,y) \in \mathcal{T}'$, and S is not a cut-set for $(G', \mathcal{T}')$. If $v \notin \{x,y\}$ but the x-y-path contains v, then we may omit it from the path to obtain a path in $G' \setminus S$, as all the neighbors of v are connected by edges in G'. Thus again S is not a cut-set for $(G', \mathcal{T}')$. Finally, if $v \in \{x,y\}$, we may assume without loss of generality, that $v = x$ and $v \neq y$ as $(v,v) \notin T$. Let u be the neighbor of v on the path. The graph $G' \setminus S$ contains the u-y-path and $(u,y) \in \mathcal{T}'$ by the construction of $\mathcal{T}'$. Hence also in this case S is not a cut-set for $(G', \mathcal{T}')$.

Now suppose that S is not a cut-set for $(G', \mathcal{T}')$. Therefore there is a pair $(x, y) \in \mathcal{T}'$ such that there is an x-y-path in $G' \setminus S$. If $(x, y) \in T$ and the path contains at most one neighbor of v, then this path is also contained in $G \setminus S$. If the path contains at least two neighbors of v, then we may go from the first neighbor of v on the path to v and from v to the last neighbor of v on the path, obtaining an x-y-path in $G \setminus S$. If $(x, y) \notin \mathcal{T}$, then assume without loss of generality that $(v, y) \in T$ and x is a neighbor of v. Furthermore, as all neighbors of v have a request to y in $\mathcal{T}'$ in this case, we may assume that the path does not contain any further neighbor of v, except for x. Now we can prolong the path by adding v to the beginning to obtain a v-y-path in $G \setminus S$. Summing up, S is not a cut-set for $(G, \mathcal{T})$, finishing the proof. $\square$

As we have said, if $(v, v) \in T$ for some $v \in V$, then v is in any cut-set. Hence we apply the following reduction rule as often as possible.

Reduction Rule 1 *If $(v, v) \in T$, then delete v from $(G, \mathcal{T})$.*

Observe now, that one can obtain an $\mathcal{O}(2^n \cdot n^{\mathcal{O}(1)})$-time algorithm for MultiCut by branching for each vertex into two branches — either the vertex is deleted or contracted. Lemmata 1 and 2 give recipe how the minimum cut-sets returned by the recursive calls need to be modified to obtain the minimum cut-set for the instance (see Function Cut).

Function Cut$(G, \mathcal{T})$

```
begin
    if $\mathcal{T} = \emptyset$ then
        return $\emptyset$;
    Let $v$ be an arbitrary vertex of $G$;
    $(G', \mathcal{T}') \leftarrow Delete(v, (G, \mathcal{T}))$;
    $S_1 \leftarrow \{v\} \cup Cut(G', \mathcal{T}')$;
    if $(v, v) \in \mathcal{T}$ then
        return $S_1$;
    $(G', \mathcal{T}') \leftarrow Contract(v, (G, \mathcal{T}))$;
    $S_2 \leftarrow Cut(G', \mathcal{T}')$;
    if $|S_1| \leq |S_2|$ then
        return $S_1$;
    else
        return $S_2$;
end
```

As our algorithm is based on the same operations, we will no longer describe how the minimum cut-set is actually obtained, for brevity. The speed up of our algorithm is achieved by carefully choosing the vertices to branch on and omitting the branches that cannot lead to a (minimum) cut-set.

To conclude this section, we show that the variant of MULTICUT, where the terminals are forbidden to delete can be reduced to MULTICUT.

Observation 1 MULTICUT WITH UNDELETABLE TERMINALS *can be reduced in polynomial to* MULTICUT *with at most the same number of vertices.*

Proof. It is enough to contract all terminals. By Lemma 2, a subset of vertices is a cut-set in the original instance not containing the terminals, if and only if it is a cut set in the resulting instance. The contraction can be clearly carried out in polynomial time and does not increase the number of vertices. □

4 Our algorithm

To guide the branching, our algorithm maintains a clique C, which we call the *active clique*. Hence, in the recursive calls, we give $G, \mathcal{T}$, and C as arguments. We should explain how our two operations affect the active clique. If $v \notin C$, then C stays untouched after the operations. If $v \in C$ and we delete v, then we let $C := C \setminus \{v\}$ whereas if we contract v we let $C := C \setminus \{v\} \cup N(v)$. Note that in the last case the new C is indeed a clique, as originally C must have been a subset of $N[v]$.

The bigger C is, the closer together are the vertices of the graph, which is beneficial for the algorithm. Hence, if the graph has a big active clique, we consider it little smaller. More precisely, we use the Measure and Conquer approach [9] and our measure μ for the size of the instance $(G, \mathcal{T}, C)$ is given by the following formula:

$$\mu = |V| - \alpha|C|,$$

where $0 < \alpha < 0.1$.

Along with the Reduction Rule 1 we apply also the following two reduction rules:

Reduction Rule 2 *If vertex v is isolated in G and Reduction Rule 1 does not apply, then contract v in $(G, \mathcal{T})$.*

Note that vertex v does not influence whether a set is a cut-set, since it always forms a component for itself and $(v, v) \notin \mathcal{T}$, justifying the correctness of the rule.

Reduction Rule 3 *If $C = \emptyset$, then pick any vertex $v \in V$ and let $C = \{v\}$.*

Note that each of the reduction rules decreases the value of μ.

We now describe the branching rules. We apply them in the given order, that is, a latter branching rule is only applied if none of the earlier ones applies. Moreover, we apply the reduction rules exhaustively before applying any of the branching rules. We argue the correctness of the rules, but postpone the discussion of the running time of the whole algorithm.

Rule 1 *If there is $(x, y) \in T$ such that the shortest path P between x and y in G is of length at most 3, then denote $V(P) = \{v_1, \ldots, v_t\}$ for some $t \in \{2, \ldots 4\}$ in such a way that $V(P) \setminus C = \{v_1, \ldots, v_s\}$ for some $s \in \{t-2, t-1, t\}$. Branch into following ways:*

- *v_1 is deleted;*
- *v_1 is contracted, v_2 is deleted;*

- *v_1, v_2 are contracted, v_3 is deleted;*

  $\vdots$
- *$v_1, \ldots, v_{t-1}$ are contracted, v_t is deleted.*

In the case described in Rule 1 at least one of the vertices $v_1, \ldots, v_t$ must be deleted and the rule explores all such options. It follows that the rule is correct.

Rule 2 *Let $N(C) = \bigcup_{c \in C} N(c) \setminus C$. If $|N(C)| \leq \frac{2}{3}|C|$ then for every $S_0 \subseteq (C \cup N(C))$ with $|S_0| \leq |N(C)|$ branch in the following way: delete all vertices in S_0 and contract all vertices in $(C \cup N(C)) \setminus S_0$.*

The correctness of the rule follows from the following lemma:

Lemma 3. *If the situation is as in Rule 2, then there is a minimal cut-set S such that $|S \cap (C \cup N(C))| \leq |N(C)|$.*

Proof. Let S be a minimal cut-set such that $|S \cap (C \cup N(C))| > |N(C)|$. We claim that $S' = (S \setminus C) \cup N(C)$ is also a cut-set, contradicting the minimality of S as $|S'| < |S|$. Suppose S' is not a cut-set. Then there is a pair $(x, y) \in T$ such that there is an x-y-path P in $G \setminus S'$. If $V(P) \cap (C \cup N(C)) = \emptyset$, then P is also present in $G \setminus S$ contradicting S being a cut-set. If $V(P) \cap (C \cup N(C)) \neq \emptyset$, then either $V(P) \cap N(C) \neq \emptyset$ or $V(P) \subseteq C$. The former case cannot appear as $V(P) \subseteq (V(G) \setminus S')$ and S' contains $N(C)$ and the later case implies that P is of length at most 1. However, in this case Rule 1 would apply, a contradiction. □

The correctness of Rule 2 now follows from Lemmata 1 and 2 as we exhaustively try all possible intersections of the minimal cut-set with $C \cup N(C)$.

Rule 3 *If there is a vertex $v \in C$ such that $|N(v) \setminus C| \geq 3$, then branch into the following canonical ways:*

- *delete v;*
- *contract v.*

As the branching explores the two canonical options, the correctness is clear. The following explains the significance of the coming rules.

Lemma 4. *If the Rules 2 and 3 do not apply, then there is a vertex v in $N(C)$ with $|N(v) \cap C| \leq 2$.*

Proof. Suppose there is no such vertex and let us count the number z of edges between C and $N(C)$ in G. Since Rule 3 does not apply, we know that $z \leq 2|C|$. On the other hand, we know that $z \geq 3|N(C)|$ as otherwise there is a vertex in $N(C)$ incident to at most two edges with the other endpoint in C. As Rule 2 does not apply, we have $3|N(C)| > 3 \cdot \frac{2}{3}|C| = 2|C|$. Hence $2|C| < 3|N(C)| \leq z \leq 2|C|$— a contradiction. □

Rule 4 *If v is a vertex in $N(C)$ with $|N(v) \cap C| \leq 2$ and $|N(v) \setminus C| \leq 3$, then denote $N(v) = \{u_1, \ldots, u_t\}$ such that $N(v) \cap C = \{u_{s+1}, \ldots, u_t\}$ for some $t \in \{1, \ldots, 5\}$ and $s \in \{t-2, t-1\}$. Branch into the following ways:*

- *u_1 is contracted;*
- *u_1 is deleted, u_2 is contracted;*

  ⋮
- *$u_1, \ldots, u_{t-1}$ are deleted, u_t is contracted;*
- *$u_1, \ldots, u_t$ are deleted, v is contracted.*

To see the correctness of this rule, observe that the first t branches correspond to one of the neighbors of v not being part of the cut-set constructed, while the last one corresponds to the whole neighborhood of v being part of the cut-set constructed. In this sense the branching is exhaustive. In the last branch the vertex v is contracted by Reduction Rule 2.

Rule 5 *If v is a vertex in $N(C)$ with $|N(v) \cap C| \leq 2$ and $|N(v) \setminus C| \geq 4$, then let $u \in C \cap N(v)$ and branch into the following ways:*

- *v is deleted;*
- *v is contracted, u is deleted;*
- *v and u are contracted.*

Since the rule only applies the canonical branching to v and then to u in one of the branches, the correctness is clear.

5 Time complexity

In this section we analyze the time complexity of our algorithm. As the algorithm is recursive, we first bound the number of recursive calls and then the time spent per each call.

Let us first bound the number of terminal calls $T(\mu)$ produced, when the algorithm is executed on an instance with measure at most μ. Recall that $\mu = |V| - \alpha|C|$ and, since $\alpha < 0.1$, we have $0.9|V| < \mu \leq |V|$. We claim that $T(\mu) \leq \lambda^{\mu}$ for $\lambda = 1.9865$, $\alpha = 0.032$, and all values of $\mu \geq 0$. We prove the claim by induction on μ.

If $\mu \leq 0$, then the graph is empty and the instance can be resolved by outputting $\emptyset$, giving one terminal call. For $0 < \mu \leq 1$, the graph contains at most one vertex and Reduction Rule 1 or 2 applies, reducing to previous case and giving one terminal call. This gives the base of the induction.

Now suppose we are facing an instance of measure μ and the claim holds for instances with measure μ' where $\mu' < \mu$. Note, that the measure is decreased by one for each vertex not in C deleted or contracted, by at least $1 - \alpha$ for each vertex in C contracted or deleted, and by α for each vertex newly put in C (e.g., due to contraction of its neighbor).

If any of the reduction rules applies to the instance, then the measure gets decreased without increasing the number of terminal calls and the claim follows. Now let us distinguish, which of the branching rules applies.

If Rule 1 applies, then in the branch i (the one where v_i is deleted) the measure is reduced by at least i if $i \le s$ and by at least $s + (1-\alpha)(i-s) = i - \alpha(i-s)$ if $i > s$. It follows that $T(\mu) \le \sum_{i=1}^{s} T(\mu - i) + \sum_{i=s+1}^{t} T(\mu - i + \alpha(i-s))$. Hence to prove the claim it is enough to prove that $\lambda^{\mu} \ge \sum_{i=1}^{s} \lambda^{\mu-i} + \sum_{i=s+1}^{t} \lambda^{\mu-i+\alpha(i-s)}$ which is equivalent to $1 \ge \sum_{i=1}^{s} \lambda^{-i} + \sum_{i=s+1}^{t} \lambda^{-i+\alpha(i-s)}$. Observe that decreasing s increases the right hand side, as $\lambda > 1$ and $\alpha > 0$. Hence, it suffices to prove the inequality for $s = t - 2$.

Distinguishing the value of t the claim follows from that

- $1 \ge \lambda^{-1+\alpha} + \lambda^{-2+2\alpha} \doteq 0.778$ ($t = 2$),
- $1 \ge \lambda^{-1} + \lambda^{-2+\alpha} + \lambda^{-3+2\alpha} \doteq 0.896$ ($t = 3$), and
- $1 \ge \lambda^{-1} + \lambda^{-2} + \lambda^{-3+\alpha} + \lambda^{-4+2\alpha} \doteq 0.954$ ($t = 4$), for $\lambda = 1.9865$ and $\alpha = 0.032$.

If Rule 2 applies, let us denote $a = |C|$, $b = |N(C)|$, $m = a + b$, and $\beta = \frac{b}{m}$. The measure drops in each case by at least $b + a(1-\alpha) = m(1-\alpha) + b\alpha$. Hence $T(\mu) \le \sum_{c=0}^{b} \binom{m}{c} T(\mu - m(1-\alpha) - b\alpha)$. To prove the claim we need to show that $1 \ge \sum_{c=0}^{b} \binom{m}{c} \lambda^{-m(1-\alpha)-b\alpha}$ for any $b \le \frac{2}{3}a$. By [10, Lemma 3.13] we have that $\sum_{c=0}^{b} \binom{m}{c} \lambda^{-m(1-\alpha)-b\alpha} \le \lambda^{-m(1-\alpha)-b\alpha} \cdot (\frac{1}{\beta})^{\beta m} (\frac{1}{1-\beta})^{(1-\beta)m} = \left(\frac{1}{\lambda^{1-\alpha}} (\frac{1}{\beta\lambda^{\alpha}})^{\beta} (\frac{1}{1-\beta})^{1-\beta} \right)^{m}$ and it remains to prove that $f(\beta) = \frac{1}{\lambda^{1-\alpha}} \cdot (\frac{1}{\beta\lambda^{\alpha}})^{\beta} \cdot (\frac{1}{1-\beta})^{1-\beta} \le 1$ for every $0 \le \beta \le \frac{\frac{2}{3}}{1+\frac{2}{3}} = \frac{2}{5}$. We first show that this function is nondecreasing on the interval $(0, \frac{2}{5}]$. To this end, consider the function $g(\beta) = \ln f(\beta) = -(1-\alpha)\ln\lambda - \beta(\ln\beta + \alpha\ln\lambda) - (1-\beta)\ln(1-\beta)$. Function g is well defined on the interval and if g is nondecreasing, then so is f. For the derivative we have $g'(\beta) = -(\ln\beta + \alpha\ln\lambda) - \frac{\beta}{\beta} + \ln(1-\beta) + \frac{1-\beta}{1-\beta} = \ln\frac{1-\beta}{\beta\lambda^{\alpha}}$. Thus, $g(\beta)$ is nondecreasing as long as $\frac{1-\beta}{\beta} \ge \lambda^{\alpha}$. But since $\frac{3}{2} > \lambda^{\alpha} \doteq 1.02$, function g and, hence, also f is non-decreasing for all $\beta \in (0, \frac{2}{5}]$. Therefore, it is enough to notice that $1 \ge f(\frac{2}{5}) = \frac{1}{\lambda^{1-\alpha}} (\frac{5}{2})^{\frac{2}{5}} (\frac{5}{3})^{\frac{3}{5}} \doteq 0.99982$ and $1 \ge f(0) = \frac{1}{\lambda^{1-\alpha}} \doteq 0.51$ for $\lambda = 1.9865$ and $\alpha = 0.032$.

If Rule 3 applies, then the measure gets reduced by 1 and at least $1 + 2\alpha$, respectively, as in the latter case v is removed from C, while its at least 3 neighbors become part of C. Hence, we have $T(\mu) \le T(\mu - 1) + T(\mu - 1 - 2\alpha)$. To prove the claim it is enough to observe that $1 \ge \lambda^{-1} + \lambda^{-1-2\alpha} \doteq 0.985$.

If Rule 4 applies, then in the branch i (the one where u_i is contracted) the measure is reduced by at least i if $i \le s$ and by at least $s + (1-\alpha)(i-s) + \alpha = i - \alpha(i-s-1)$ if $t \ge i > s$, as in this case v becomes part of C, whereas in the last branch it is reduced by $s+(1-\alpha)(t-s)+1 = t-\alpha(t-s)+1$. It follows that $T(\mu) \leq \sum_{i=1}^{s} T(\mu-i) + \sum_{i=s+1}^{t} T(\mu-i+\alpha(i-s-1)) + T(\mu-t+\alpha(t-s)+1)$. Hence to prove the claim it is enough to prove that $1 \geq \sum_{i=1}^{s} \lambda^{-i} + \sum_{i=s+1}^{t} \lambda^{-i+\alpha(i-s-1)} + \lambda^{-t+\alpha(t-s)-1}$. Observe that decreasing s increases the right hand side, as $\lambda > 1$ and $\alpha > 0$. Hence, it suffices to prove the inequality for $s = t-2$.

Distinguishing the value of t we have that

- $1 \geq \lambda^{-1} + \lambda^{-2+\alpha} \doteq 0.762$ $(t=1)$,
- $1 \geq \lambda^{-1} + \lambda^{-2+\alpha} + \lambda^{-3+2\alpha} \doteq 0.896$ $(t=2)$,
- $1 \geq \lambda^{-1} + \lambda^{-2} + \lambda^{-3+\alpha} + \lambda^{-4+2\alpha} \doteq 0.954$ $(t=3)$,
- $1 \geq \lambda^{-1} + \lambda^{-2} + \lambda^{-3} + \lambda^{-4+\alpha} + \lambda^{-5+2\alpha} \doteq 0.984$ $(t=4)$,
- $1 \geq \lambda^{-1} + \lambda^{-2} + \lambda^{-3} + \lambda^{-4} + \lambda^{-5+\alpha} + \lambda^{-6+2\alpha} \doteq 0.9986$ $(t=5)$, for $\lambda = 1.9865$ and $\alpha = 0.032$.

Finally, if Rule 5 applies, then the measure is decreased by 1, by $2-\alpha$, and by at least $2+3\alpha$, respectively, as in the last branch the neighbors of v outside C become part of C, but u is removed from C. Therefore, we have $T(\mu) \leq T(\mu-1) + T(\mu-2+\alpha) + T(\mu-2-4\alpha)$. To prove the claim it is enough to observe that $1 \geq \lambda^{-1} + \lambda^{-2+\alpha} + \lambda^{-2-3\alpha} \doteq 0.99968$.

Now to compute the total number of recursive call, observe that each branching rule reduces the number of vertices in the graph in each branch, and, hence, the total number of recursive calls is at most $n \cdot \lambda^{\mu}$. In each recursive call we first apply the reduction rules exhaustively and then check which of the branching rules applies. Reduction Rule 1 can be applied to each relevant vertex in $\mathcal{O}(n)$, without creating new opportunities to apply it. Therefore, it can be applied exhaustively in $\mathcal{O}(n^2)$ time. Similarly, Reduction Rule 2 can be applied to each vertex in $\mathcal{O}(n)$ time and this does not create any new opportunities to apply this rule or the previous one. Reduction Rule 3 can be always applied in constant time.

To apply Rule 1 we compute the distance between every pair of vertices in $\mathcal{O}(n^3)$ time and then check for each pair $(u,v) \in \mathcal{T}$ their distance. To delete a vertex from a graph takes $\mathcal{O}(n)$ time whereas contracting a vertex takes $\mathcal{O}(n^2)$. Thus, the whole preparation of the graph for each branch takes $\mathcal{O}(n^2)$ time in this case.

The size of the neighborhood $N(C)$ can be computed in $\mathcal{O}(n^2)$ time. It follows from the above ideas that a graph can be prepared for each branch in $\mathcal{O}(n^3)$ time. Therefore, Rule 2 can be applied in $\mathcal{O}(n^3)$ time, accounting the time for the preparation of the graph to the call executed.

Rules 3–5 can be applied in $\mathcal{O}(n^2)$ time, since we only have to compute for each vertex the number of its neighbors in C and outside C and then prepare graphs for constant number of branches, each deleting or contracting only constant number of vertices.

Altogether we spend only $\mathcal{O}(n^3)$ time per a recursive call. Since μ is always at most n we obtain $\mathcal{O}(1.9865^n \cdot n^4) = \mathcal{O}(1.987^n)$ running time for the whole algorithm. We only need a polynomial space. This completes the proof of our theorem.

6 Conclusions

In this paper we gave an algorithm for Multicut running in time $\mathcal{O}(1.987^n)$, the first algorithm breaking the barrier of 2^n. One can obtain an algorithm for edge variant of Multicut running in time $2^n n^{\mathcal{O}(1)}$. It is an interesting problem to obtain an algorithm for Edge Multicut running in time $(2-\epsilon)^n n^{\mathcal{O}(1)}$ for some fixed $\epsilon > 0$. Finally, it would also be interesting to obtain an algorithm for Multicut in directed graphs running in time $(2-\epsilon)^n n^{\mathcal{O}(1)}$ for some fixed $\epsilon > 0$.

References

1. Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is fpt. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 459–468, New York, NY, USA, 2011. ACM. 2
2. Gruia Calinescu, Cristina G. Fernandes, and Bruce Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. *Journal of Algorithms*, 48(2):333 – 359, 2003. 1
3. Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *computational complexity*, 15(2):94–114, 2006. 2
4. Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. 2
5. Rajesh Chitnis, FedorV. Fomin, Daniel Lokshtanov, Pranabendu Misra, M.S. Ramanujan, and Saket Saurabh. Faster exact algorithms for some terminal set problems. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation*, volume 8246 of *Lecture Notes in Computer Science*, pages 150–162. Springer International Publishing, 2013. 2
6. Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In *SODA*, pages 1713–1725, 2012. 2
7. Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. IPEC, 2011. 2
8. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23:864–894, 1994. 1, 2
9. Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, August 2009. 5
10. Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2010. 8
11. FedorV. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014. 2
12. N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. 2

13. Yoichi Iwata. A faster algorithm for dominating set analyzed by the potential method. In *Proceedings of the 6th International Conference on Parameterized and Exact Computation*, IPEC'11, pages 41–54, Berlin, Heidelberg, 2012. Springer-Verlag. 2
14. Dániel Marx. Parameterized graph separation problems. *Theoret. Comput. Sci.*, 351(3):394–406, 2006. 2
15. Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *STOC*, pages 469–478, 2011. 2
16. Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Paths, flowers and vertex cover. In *ESA*, pages 382–393, 2011. 2
17. Igor Razgon and Barry O'Sullivan. Almost 2-sat is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009. 2
18. J.M Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425 – 440, 1986. 2
19. Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for undirected feedback vertex set. In Peter Widmayer, Yinfeng Xu, and Binhai Zhu, editors, *Combinatorial Optimization and Applications*, volume 8287 of *Lecture Notes in Computer Science*, pages 153–164. Springer International Publishing, 2013. 2