HITTING SELECTED (ODD) CYCLES [*]

DANIEL LOKSHTANOV[†], PRANABENDU MISRA[‡], M. S. RAMANUJAN[†], AND SAKET SAURABH [‡]

Abstract. In the Subset Odd Cycle Transversal (Subset OCT) problem, the input is a graph G, a subset of vertices T, a positive integer k and the objective is to determine whether there exists a k-sized vertex subset that intersects every odd cycle containing a vertex from T. Clearly, Subset OCT is a generalization of the classic Odd Cycle Transversal problem where the objective is to determine whether there exists a k-sized vertex subset that intersects every odd cycle in the given graph. We remark that Subset OCT also generalizes the well known Multiway Cut problem, as well as a parity constrained variant, the Odd Multiway Cut problem. Recently, Kakimura et al. (SODA 2012) proposed a fixed parameter tractable (FPT) algorithm for this problem that runs in time $f(k)mn^3$ using the theory of graph minors, where f is some function, and n and m denote the number of vertices and edges in the graph. However, the dependence of this function on k is at least triple exponential.

Key words. Algorithms and Data Structures. Graph Algorithms. Parameterized Algorithms.

AMS subject classifications. 68W40. 68Q25. 68R10.

1. Introduction.

In a *covering* or *transversal problem* we are given a universe of elements U, a family $\mathcal{F}$ ($\mathcal{F}$ could be given implicitly) and an integer k and the objective is to check whether there exists a subset of U of size at most k which intersects all the elements of $\mathcal{F}$. Several natural problems on graphs can be framed in the form of such a problem. For instance, consider the classic Odd Cycle Transversal (OCT) problem. Here, given a graph G and a positive integer k, the objective is to decide whether there exists a vertex subset S (also called a transversal) of size at most k which intersects all odd cycles, that is, $G \setminus S$ is a bipartite graph. The OCT problem is a covering problem with the vertex set $V(G)$ being U and $\mathcal{F}$ being the set of odd cycles. Similarly, (Directed) Feedback Vertex Set – given an undirected (directed) graph G and a positive integer k decide whether there is vertex subset of size at most k that intersects all cycles (directed cycles) – is also a covering problem.

Recently, a natural generalization of covering problems has attracted a lot of attention from the point of view of parameterized complexity. In this generalization, apart from U, $\mathcal{F}$ and k, we are also given a subset T of U and the objective is to decide whether there is a subset of U of size at most k that intersects all the sets in $\mathcal{F}$ that contain an element in T. This leads to the *subset variant* of classic covering problems; typical examples include Subset Feedback Vertex Set (Subset FVS), Subset Directed Feedback Vertex Set or Subset Odd Cycle Transversal (Subset OCT). All these problems have received considerable attention and they have been shown to be *fixed-parameter tractable* (FPT) [8, 4, 15]. Formally, a *parameterization* of a problem is the assignment of an integer k to each input instance and we say that a parameterized problem is FPT if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the size of the input instance and f is an arbitrary computable function depending only on the parameter k. For more background, the

[*] Submitted to the editors DATE.

Funding:

[†]Department of Informatics, University of Bergen, Norway. E-mail: daniello@ii.uib.no,ramanujan@ac.tuwien.ac.at

[‡]The Institute of Mathematical Sciences, HBNI, Chennai, India E-mail: {pranabendu, saket}@imsc.res.in

reader is referred to the monographs [11, 27, 6].

In 2011, Cygan et al. [8] and Kawarabayashi et al. [16] independently showed that Subset FVS is FPT. Following that, Chitnis et al. [4] showed that the directed variant of the same problem, Subset Directed Feedback Vertex Set is FPT. Kakimura et al. [15] initiated the study of Subset OCT and proposed an FPT algorithm for this problem using the theory of parity version of graph minors, that runs in time $f(k)mn^3$ where f is some function, and n and m denote the number of vertices and edges in the graph. The Subset OCT problem is formally defined as follows.

Subset OCT

Instance :	A graph $G = (V, E)$ on n vertices and m edges, a subset of vertices T, and a positive integer k.
Parameter :	k
Question :	Is there a subset of k vertices that intersects every odd cycle that contains a vertex of T ?

However, the dependence of the algorithm of Kakimura et al. on the parameter k is at least triple exponential. In this paper, we give an algorithm which avoids the use of the theory of graph minors, is self contained, and improves this dependence to a single exponential function of a polynomial in k. Our algorithm utilizes a recursive application of "generalized" important separators introduced in [20] to reduce the subset version of this problem to the standard version of the problem. In particular, we show how to reduce the given problem to a special case of the problem in time $2^{k^{\mathcal{O}(1)}}n^{\mathcal{O}(1)}$. More precisely, we prove the following theorem.

Theorem 1.1. *There is an algorithm that, given an instance $(G = (V, E), T, k)$ of* Subset OCT *runs in time $2^{\mathcal{O}(k^3 \log k)}mn^2 \log^2 n$ and either returns a solution for this instance or concludes correctly that one does not exist, where $n = |V|$ and $m = |E|$.*

In the statement of the theorem above, *solution* refers to a set of at most k vertices that intersects every odd cycle of G that contains a vertex of T. This is the first FPT algorithm for this problem where the exponential dependence of the running time of the algorithm on k is polynomial. The algorithm given by Theorem 1.1 improves upon the algorithm of Kakimura et al. with respect to both the parameter as well as the input size. Subset OCT is central to covering, as well as graph separation problems as Subset OCT generalizes OCT, the well known Multiway Cut problem, as well as a parity constrained variant, the Odd Multiway Cut problem. Thus, our approach has to build on techniques from both worlds.

Related Results on OCT*.*. The parameterized complexity of OCT was a well known open problem for a long time. In 2003, in a breakthrough paper, Reed et al. [31] showed that OCT is FPT by developing an algorithm for the problem running in time $\mathcal{O}(3^k kmn)$. We use n and m to denote the number of vertices and edges of the input graph respectively. In fact this was the first time that the iterative compression technique was used. This technique has been useful in resolving several other open problems in the area of parameterized complexity, including Almost-2-SAT and Directed Feedback Vertex Set. However, the algorithm for OCT had seen no further improvements in the last 9 years, though several reinterpretations of the algorithm have been published [13, 22]. Only recently, Lokshtanov et al. [19] obtained a faster algorithm for the problem running in time $\mathcal{O}^*(2.32^k)$ using a branching based on linear programming. We write $\mathcal{O}^*(f(k))$ for a time complexity of the form $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$, where $f(k)$ grows exponentially with k. There is another theme of research in parameterized complexity where the objective is to minimize the dependence of n at the cost of a slow growing function of k. It is very hard to get an FPT algorithm which runs faster than $\mathcal{O}(n^2)$ or $\mathcal{O}(mn)$, using the iterative compression method. Kawarabayashi and Reed [17] overcame this difficulty and obtained an algorithm that runs in time $\mathcal{O}(f(k)m\cdot\alpha(m,n))$ using tools from graph minors and odd variants of graph minors. Here the function $\alpha(m,n)$ is the inverse of the Ackermann function (see the book by Tarjan [32]). In another result Fiorini et al. [10] showed, that there is a $\mathcal{O}(2^{\mathcal{O}(k^6)}n)$ time algorithm for Odd Cycle Transversal when the input graph is restricted to a planar graph. Other results in this direction include a $\mathcal{O}(2^{\mathcal{O}(k\log k)}n)$ algorithm for OCT on planar graphs [23] which was later improved by the first linear time FPT algorithms for OCT on general graphs, both of which run in time $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m+n))$ [29, 14].

Related work in graph separation problems. Marx [24] was the first to consider cut problems in the context of parameterized complexity. He observed that Multiway Cut can be shown to be FPT by a simple application of graph minors, (see [24, Section 3]). He also gave an algorithm for Multiway Cut with a running time of $\mathcal{O}^*(4^{k^3})$. The current fastest algorithm for Multiway Cut runs in time $\mathcal{O}^*(2^k)$ [7]. The notions used in this paper have been useful in settling the parameterized complexity of a number of problems including Directed Feedback vertex Set [3], Almost 2-SAT [30] and Above Guarantee Vertex Cover [30, 28]. In addition, Marx and Razgon [26] used these notions to show that Multicut, finding k vertices to disconnect given pairs of terminals is FPT, which was also independently proved by Bousquet, Daligault and Thomassé [1]. Subsequent FPT results which use these ideas include those for Directed Multiway Cut [5], Directed Subset Feedback Vertex Set [4], Multicut on DAGs [18], and Parity Multiway Cut [20]. Recently, a randomized $\mathcal{O}(c^k\cdot(n+m))$ time and a deterministic $\mathcal{O}(k^{\mathcal{O}(k)}\cdot(n+m))$ time algorithms for Subset Feedback Vertex Set have also been designed [21].

2. Preliminaries.

Given a path P (cycle C), we refer to the number of edges in P (C) as the length of P (C) and denote it by $|P|$ (respectively $|C|$). We call a path (cycle) an odd (even) path if the length of the path (cycle) is odd (respectively even). We refer to the parity of $|P|$ ($|C|$) as the parity of the path P (cycle C). If P is a path from some vertex in a set X to some vertex in a set Y such that its internal vertices are disjoint from $X\cup Y$, we say that P is an X-Y path. Let $\lambda_G(X,Y)$ denote the number of vertex disjoint paths between X and Y in the graph G. If X contains a single vertex x, we say that P is an x-Y path. Given a graph $G=(V,E)$ and $T\subseteq V$, paths with only the end points in T are called *T-paths* and if their length is odd (even), then *odd (even)* T-paths. Similarly, cycles which intersect a set T are called *T-cycles* and if their length is odd (even), then *odd (even)* T-cycles. If a cycle intersects vertices from two sets S and T, then it is called an S-T cycle. A set intersecting all odd cycles is called an oct and a set intersecting all odd T-cycles is called a T-oct. A set of paths are said to be internally vertex disjoint if they do not intersect except possibly at their endpoints. For an instance (G,T,k) of Subset OCT, the vertices in the set T are called *terminals*.

2.1. Block Decomposition.

A *cut vertex* of a graph G is a vertex whose removal leads to a graph with strictly more connected components. Similarly, a *bridge* of a graph G is an edge whose removal leads to a graph with strictly more connected components.

Definition 2.1. ([9]) *A maximal connected subgraph without a cut vertex is called a* ***block****. Every block of a graph G is either a maximal 2-connected subgraph, or a bridge or an isolated vertex.*

By maximality, distinct blocks of G overlap in at most one vertex, which is then a cut vertex of G. Therefore, every edge of G lies in a unique block and G is the union of its blocks.

Definition 2.2. *Let A denote the set of cut vertices of G and B the set of its blocks. The bipartite graph on $A \cup B$ where $a \in A$ and $b \in B$ are adjacent precisely when $a \in b$ is called the block graph of G.*

Proposition 2.3. ([9]) *The block graph of a connected graph is a tree.*

2.2. Separators and almost separators. Important Separators. The notion of important separators was formally introduced in [24] to handle the Multiway Cut problem and the same concept was used implicitly in [2] to give an improved algorithm for the same problem. In this subsection, we recall some definitions related to important separators and a few lemmas which will be required for our algorithm.

Definition 2.4. *Let $G = (V, E)$ be an undirected graph, let $X, S \subseteq V$ be vertex subsets. We denote by $R_G(X, S)$ the set of vertices of $V \setminus S$ reachable from X in the graph $G \setminus S$ and we denote by $NR_G(X, S)$ the set of vertices of $G \setminus S$ which are not reachable from X in the graph $G \setminus S$. We drop the subscript G if it is clear from the context.*

Definition 2.5. *Let $G = (V, E)$ be an undirected graph and let $X, Y \subset V$ be two disjoint vertex sets. A subset $S \subseteq V \setminus (X \cup Y)$ is called an X-Y* **separator** *in G if $R_G(X, S) \cap Y = \emptyset$ or in other words there is no path from X to Y in the graph $G \setminus S$. We denote by $\lambda_G(X, Y)$ the size of the smallest X-Y separator in G. An X-Y separator S_1 is said to* ***cover*** *an X-Y separator S with respect to X if $R(X, S_1) \supset R(X, S)$ and S_1 is said to* ***dominate*** *S if it covers S and $|S_1| \leq |S|$. If the set X is clear from the context, we just say that S_1 dominates S. An X-Y separator is said to be inclusion wise minimal if none of its proper subsets is an X-Y separator.*

Definition 2.6. *Two X-Y separators S and S_1 are said to be* ***incomparable*** *if neither covers the other.*

Observation 2.7. *Let S_1 and S_2 be two incomparable X-Y separators. Then, $R(X, S_1) \cap S_2 \neq \emptyset$ and $R(X, S_2) \cap S_1 \neq \emptyset$. That is, there is a vertex of S_1 reachable from X in the graph $G \setminus S_2$ and a vertex of S_2 reachable from X in the graph $G \setminus S_1$. Also, $NR(X, S_1) \cap S_2 \neq \emptyset$ and $NR(X, S_2) \cap S_1 \neq \emptyset$. That is, there is a vertex of S_1 separated from X in the graph $G \setminus S_2$ and a vertex of S_2 separated from X in the graph $G \setminus S_1$.*

Definition 2.8. *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be vertex sets and $S \subseteq V$ be an X-Y separator in G. We say that S is an* ***important*** *X-Y separator if it is inclusion-wise minimal and there does not exist another X-Y separator S_1 such that S_1 dominates S with respect to X.*

Lemma 2.9. ([24]) *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets. There exists a unique important X-Y separator S^* of size $\lambda_G(X, Y)$ and it can be computed in time $\mathcal{O}(\ell(m + n))$ where $\ell = \lambda(X, Y)$.*

Lemma 2.10. ([2, 26]) *The number of important X-Y separators of size at most k is at most 4^k and these can be enumerated in time $\mathcal{O}(4^k k(m + n))$ and any vertex v*

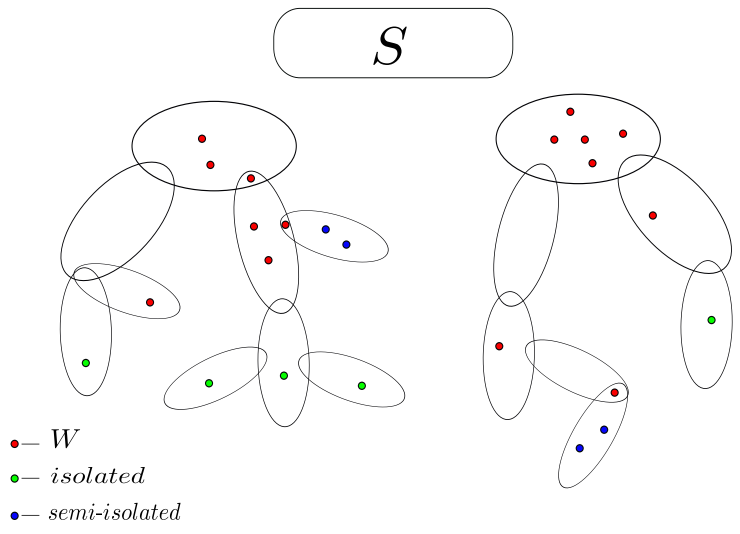

Fig. 1. *An illustration of vertices isolated and semi-isolated from the set W in $G \setminus S$.*

appears in at most 4^k sets which induce a connected subgraph and whose neighborhood is an important v-X separator of size at most k.

Definition 2.11. *Given sets X and Y and a pair $(S, p(S))$ where S is a set of vertices and $p(S)$ denotes either the empty set or a vertex disjoint from S, we say that $(S, p(S))$ is an* **almost X-Y separator** *if $S \cup p(S)$ is a minimal X-Y separator. The size of an almost X-Y separator $S \cup p(S)$ is $|S \cup p(S)|$.*

Observe that for any minimal X-Y separator S, the pair $(S, \emptyset)$ is an almost X-Y separator. We now define a notion of *isolated* and *semi-isolated* vertices. These notions are frequently used in our algorithms and we will describe their uses in the subsequent subsections.

Definition 2.12. *Given a graph $G = (V, E)$ and disjoint vertex sets X and Y, we denote by $f_G(X, Y)$ the maximum number of internally vertex disjoint X-Y paths in G.*

Definition 2.13. *Let $G = (V, E)$ be a graph and let S and W be disjoint vertex sets of G (see Figure 1).*

- *A vertex $v \in V \setminus S$ is said to be* **isolated** *from W in $G \setminus S$ if v is not adjacent to W and $f_{G \setminus S}(v, W) \leq 1$.*
- *A vertex $v \in V \setminus S$ is said to be* **semi-isolated** *from W in $G \setminus S$ if*
  - *$f_{G \setminus S}(v, W) = 1$ and v is adjacent to some $x \in W$ or*
  - *if $f_{G \setminus S}(v, W) \geq 2$ and $f_{G \setminus (S \cup \{x\})}(v, W \setminus \{x\}) = 0$ for some $x \in W$.*

  This unique vertex of W is called the ***pivot*** *between the vertex v and W in $G \setminus S$ and is denoted by $\mathsf{piv}_{G \setminus S}(v, W)$.*

Any vertex in $V \setminus S$ which is neither isolated nor semi-isolated in $G \setminus S$ is called non-isolated (from W, in $G \setminus S$). In the above notations, we drop the reference to the sets S or W if it is clear from the context.

We also need the following generalization of the notion of a semi-isolated vertex to a semi isolated set.

Definition 2.14. *Let $G = (V, E)$ be a graph and S, X and Y be disjoint vertex sets. If the vertices in Y occur in a single block of $G \setminus S$ and every vertex in Y is semi-isolated from X in $G \setminus S$, we say that the set Y is semi-isolated from X in $G \setminus S$. The pivot between the sets Y and X is defined as the vertex of X which occurs as the pivot between every vertex in Y and X and is denoted by $\mathsf{piv}_{G\setminus S}(Y, X)$.*

3. Overview of the algorithm. In this section we give an overview of the algorithm and briefly describe the various steps involved in the algorithm. During this description, we state the lemmata which capture each step of our algorithm and their proofs will appear later as part of the detailed description of our algorithm. However, in the detailed description, for the sake of continuity, we will restate each lemma before giving its proof. At the end of this section, the reader will find a flowchart summarizing the sequence of high level steps executed by the algorithm.

3.1. Iterative Compression Step. Given an instance $(G = (V, E), T, k)$ of Subset OCT, where $V = \{v_1, \ldots, v_n\}$, we define a graph $G_i = G[V_i]$ where $V_i = \{v_1, \ldots, v_i\}$. We iterate through the instances $(G_i, T_i = (T \cap V_i), k)$ starting from $i = k + 1$ and for the i^{th} instance, with the help of a *known* solution $\hat{S}_i$ of size at most $k + 1$ we try to find a solution S_i of size at most k. Formally, the *compression* problem we address is the following.

Subset OCT Compression **Parameter:** k
Input: $(G = (V, E), T, k, \hat{S})$ where G is an undirected graph, T is a vertex set, k a positive integer and $\hat{S}$, a T-oct of size at most $k + 1$.
Question: Does there exist a T-oct of size at most k for this instance?

We will reduce the Subset OCT problem to at most n instances of the Subset OCT Compression problem as follows. Let $I_i = (G_i, (T \cap V_i), k, \hat{S}_i)$ be the i^{th} instance of Subset OCT Compression. Clearly, the set V_{k+1} is a solution of size at most $k + 1$ for the instance I_{k+1}. It is also easy to see that if S_{i-1} is a solution of size at most k for instance I_{i-1}, then the set $S_{i-1} \cup \{v_i\}$ is a solution of size at most $k + 1$ for the instance I_i. We use these two observations to start off the iteration with the instance $(G_{k+1}, (T \cap V_{k+1}), k, \hat{S}_{k+1} = V_{k+1})$ and try to compute a solution of size at most k for this instance. If there is such a solution S_{k+1}, we set $\hat{S}_{k+2} = S_{k+1} \cup \{v_{k+2}\}$ and try to compute a solution of size at most k for the instance I_{k+2} and so on. If, during any iteration, the corresponding instance is found to not have a solution of the required size, then it implies that the original instance is also a No instance. Finally the solution for the original input instance will be S_n. Since there can be at most n iterations, the total time taken is bounded by n times the time required to solve the Subset OCT Compression problem.

3.2. Reduction to disjoint version. We now discuss how to solve the Subset OCT Compression problem by reducing it to a bounded number of instances of the following problem.

Disjoint Subset OCT Compression **Parameter:** k
Input: $(G = (V, E), T, W, k)$ where G is an undirected graph, T is a vertex set, k a positive integer and W, a T-oct of size at most $k + 1$.
Question: Does there exist a T-oct of size at most k for this instance disjoint from W given that there is no T-oct of size at most k intersecting W?

We fix a solution S for the instance of Subset OCT Compression which maximizes the intersection with the larger solution $\hat{S}$. We guess the set $Y = S \cap \hat{S}$, delete these vertices and reduce k by $|Y|$. For each guess of Y, we set $W = \hat{S} \setminus Y$ and we solve Disjoint Subset OCT Compression on the instance $(G \setminus Y, T \setminus Y, W, k-|Y|)$, and if for some guess the answer is Yes, we return the solution for the instance corresponding to that guess. The number of guesses is bounded by $2^{|\hat{S}|} = 2^{\mathcal{O}(k)}$. Hence, the time to solve Subset OCT Compression is bounded by $2^{\mathcal{O}(k)}$ times the time to solve Disjoint Subset OCT Compression. Since we chose S such that we maximize its intersection with $\hat{S}$, we also include in the definition of the Disjoint Subset OCT Compression problem that the instance (G, T, W, k) comes with a guarantee that *there is no solution of size at most k that intersects W*. This assumption is used crucially in several proofs. The rest of the paper is devoted to proving the following lemma which, following our discussions above, proves Theorem 1.1.

Lemma 3.1. *There is an algorithm that given an instance $I = (G, T, W, k)$ of* Disjoint Subset OCT Compression*, runs in time $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$, and either computes a solution for the given instance, or correctly concludes that no solution exists.*

3.3. Reduction to 2-Connected Case. We assume without loss of generality that the graph G in any given instance of Disjoint Subset OCT Compression is connected. However, this graph may not be 2-connected. Therefore, in this step, we reduce the given instance (G, T, W, k) of Disjoint Subset OCT Compression to an equivalent instance (G', T', W', k') of Disjoint Subset OCT Compression where the graph G' is 2-connected. This property of G' will be used crucially in latter parts of our algorithm. This reduction can be found in Section 4.

3.4. Break up into special and non-special instances. Let $I = (G = (V, E), T, W, k)$ be an instance of Disjoint Subset OCT Compression where G is a 2-connected graph and let S be a hypothetical solution. Recall that the vertices in the set T are called *terminals*. Our algorithm is based on "guessing" the structure of W in $G \setminus S$. More precisely, we split our algorithm based on the following cases:

- The vertices of W belong to the same block in $G \setminus S$, for some solution S. We call such an S a *special solution* and instances having a special solution, *special instances*.
- The vertices of W do not belong to the same block in $G \setminus S$, for any solution S. We call such an S a *non-special solution* and instances haveing *only* a non-special solution, *non-special instances*

3.4.1. Handling special instances. In this case we try to find a special solution S (if exists). That is, a solution S such that the vertices of W belong to a unique block of $G \setminus S$. Again special instances can be of two types. We say that S (I) is a special solution (respectively instance) of **Type 1** if the block of $G \setminus S$ containing W does not contain a terminal. Similarly, we say that S (respectively I) is a special solution (respectively instance) of **Type 2** if the block of $G \setminus S$ containing W contains a terminal.

Type 1:. Observe that we can assume that $|T| \geq k + 1$. This is because if the number of terminals is at most k then we can return the terminal set itself as a solution. As a result, any set of $k + 1$ terminals contains a terminal vertex u which is not in S and is isolated or semi-isolated from W in $G \setminus S$. We will later show (Lemma 5.1 and Lemma 5.17) that, given any vertex which we know to be isolated or semi-isolated from W in $G \setminus S$, one can always, in time $2^{\mathcal{O}(k^2 \log k)} mn \log n$ compute a set of $2^{\mathcal{O}(k^2)}$ vertices which intersect a special solution for this instance. Hence, once we locate the vertex u, we will employ these algorithms to compute a set of $2^{\mathcal{O}(k^2)}$ vertices and recurse by branching on this set. This gives us the following lemma.

LEMMA 3.2. *There is an algorithm that, given an instance (G, T, W, k) of* DISJOINT SUBSET OCT COMPRESSION, *runs in time $2^{\mathcal{O}(k^3)} m \log n$ and either returns a solution for the given instance or correctly concludes that no Type 1 solution exists.*

Type 2:. In this case, we show that if S is *not* already an *odd cycle transversal* of G, then in time $2^{\mathcal{O}(k^2 \log k)} mn \log n$, we can locate a vertex which is isolated or semi-isolated from W in $G \setminus S$, on which we then use the same procedure as that described for the Type 1 case.

We show this by first observing that any odd cycle disjoint from S has at most one vertex in common with the block of $G \setminus S$ containing W and the rest of the vertices must therefore be isolated or semi-isolated from W. Therefore, locating *any* odd cycle disjoint from S is sufficient to find an isolated or a semi-isolated vertex. However, finding such an odd cycle is not easy and for this, we revisit the notion of generalized important separators and using this, show that we can enumerate a set of $2^{\mathcal{O}(k^2 \log k)}$ connected subgraphs and 2-connected subgraphs each of which contains an odd cycle such that at least one of these odd cycles is disjoint from S and at this point, we simply return an arbitrary pair of vertices from an odd cycle from each of these subgraphs and conclude that the set thus returned contains an isolated or semi-isolated vertex. Observe that if we cannot find such a set, then it must be the case that either there is no solution for the given instance or that there is a solution which is an odd cycle transversal for the graph G. This leads us to the following lemma.

LEMMA 3.3. *There is an algorithm that, given an instance (G, T, W, k) of* DISJOINT SUBSET OCT COMPRESSION, *runs in time $2^{\mathcal{O}(k^2 \log k)} mn \log n$ and returns a set of $2^{\mathcal{O}(k^2 \log k)}$ vertices which contain an isolated or semi-isolated with respect to some solution or concludes correctly either that the given instance has a solution which is also an odd cycle transversal for G or that the given instance has no solution.*

Finally, the fact that testing for the presence of an odd cycle transversal of size at most k can be done in time $\mathcal{O}(4^{k+\mathcal{O}(\log k)}(m+n))$ [29, 14] combined with Lemma 3.3 leads to the following lemma handling all special instances of Type 2.

LEMMA 3.4. *There is an algorithm that, given an instance (G, T, W, k) of* DISJOINT SUBSET OCT COMPRESSION, *runs in time $2^{\mathcal{O}(k^3 \log k)} mn \log n$ and either returns a solution for the given instance or correctly concludes that no Type 2 solution exists.*

3.4.2. Handling non-special instances. We now consider the case when the given instance (G, T, W, k) of DISJOINT SUBSET OCT COMPRESSION has a solution S such that there is *no* block in $G \setminus S$ which contains all the vertices in W. In this case, we first guess a partition $W = W_1 \uplus W_2$ such that the vertices in W_1 occur in a block, say B in $G \setminus S$. For a correct choice of the block and the set W_1, we will be able to show that there is a set $X \subseteq S$ which, along with at most one other vertex in G, say p, separates W_1 from W_2, that is, there is no path from W_1 to W_2 in $G \setminus (X \cup \{p\})$. Observe that this is the same as saying that W_1 is semi-isolated from $W_2 \cup p$ if $p \notin W_1$ and $W_1 \setminus p$ is semi-isolated from $W_2 \cup p$ otherwise. Furthermore, we will also distinguish between the case when the vertex p is known to us and the case when it is unknown. As a result, we now consider the following exhaustive cases.

Case (a): p is known. In this case, we show that the the following lemma holds.

LEMMA 3.5. *Given a* YES *instance* (G, T, W, k) *of* DISJOINT SUBSET OCT COMPRESSION*, let* S *be a solution for this instance. Let* $W_1 \uplus W_2$ *be a partition of* W *such that* W_1 *occurs in a block and there is a set* $X \subseteq S$ *which, along with at most one other vertex which is disjoint from* $S \cup W$*, say* p*, separates* W_1 *from* W_2*. There is an algorithm that, given* W_1*,* p *and* W_2*, runs in time* $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$ *and returns a set of* $2^{\mathcal{O}(k^2)}$ *vertices which intersects a solution for the given instance.*

The above lemma says that if, along with the information that *some vertex* intersects all W_1-W_2 paths in $G \setminus S$, we also *know precisely which vertex this is*, then we can find a set of $2^{\mathcal{O}(k^2)}$ vertices to branch on. However, this need not always be the case. It is possible that we only know that some vertex intersects all W_1-W_2 paths in $G \setminus S$ but do not know *which* vertex this is. Guessing this vertex would be too costly. As a result, we need to come up with another subroutine to handle this next case.

Case (b): Suppose that p is unknown. In this case, we will closely follow the strategy of Lemma 3.5 but with minor modifications, to prove the following lemma.

LEMMA 3.6. *Given a* YES *instance* (G, T, W, k) *of* DISJOINT SUBSET OCT COMPRESSION*, let* S *be a solution for this instance. Let* $W_1 \uplus W_2$ *be a partition of* W *such that* W_1 *occurs in a block in* $G \setminus S$ *and there is a set* $X \subseteq S$ *which, along with at most one other vertex which is disjoint from* $S \cup W$*, say* p*, separates* W_1 *from* W_2*. There is an algorithm that, given* W_1 *and* W_2*, runs in time* $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$ *and returns a set of* $2^{\mathcal{O}(k^2)}$ *vertices which*

- *intersects some solution for the given instance, or*
- *contains* p *or*
- *contains a vertex* v *which is isolated or semi-isolated with respect to some solution for the given instance.*

Now, this lemma contains in its output a vertex from which we can find a set to branch on using a previously stated lemma *or*, the previously unknown vertex which intersects all W_1-W_2 paths in $G \setminus S$. Though guessing this vertex from the entire vertex set of G is too costly, we can indeed guess this vertex from the set of bounded size returned by this lemma. As a result, we will then fall into the earlier case. To sum up, we combine Lemma 3.6 and Lemma 3.5 with (the yet to be formally stated lemmas) Lemma 5.1 and Lemma 5.17 to get the following lemma, using which one can solve non-special instances of DISJOINT SUBSET OCT COMPRESSION.

LEMMA 3.7. *Given a* YES *instance* (G, T, W, k) *of* DISJOINT SUBSET OCT COMPRESSION*, let* S *be a solution for this instance. Let* $W_1 \uplus W_2$ *be a partition of* W *such that* W_1 *occurs in a block in* $G \setminus S$ *and there is a set* $X \subseteq S$ *which, along with at most one other vertex which is disjoint from* $S \cup W$*, say* p*, separates* W_1 *from* W_2*. There is an algorithm that, given* W_1 *and* W_2*, runs in time* $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$ *and returns a set of* $2^{\mathcal{O}(k^2)}$ *vertices which intersects a solution for the given instance.*

3.5. Proof of Lemma 3.1. We are now ready to give the complete algorithm to solve DISJOINT SUBSET OCT COMPRESSION on general instances using the results described above.

Proof of Lemma 3.1. In the base case, if there is a special solution for the given instance, then we can apply Lemma 3.2 or Lemma 3.4 to compute such a solution. Therefore, we assume that there is no special solution for the given instance.

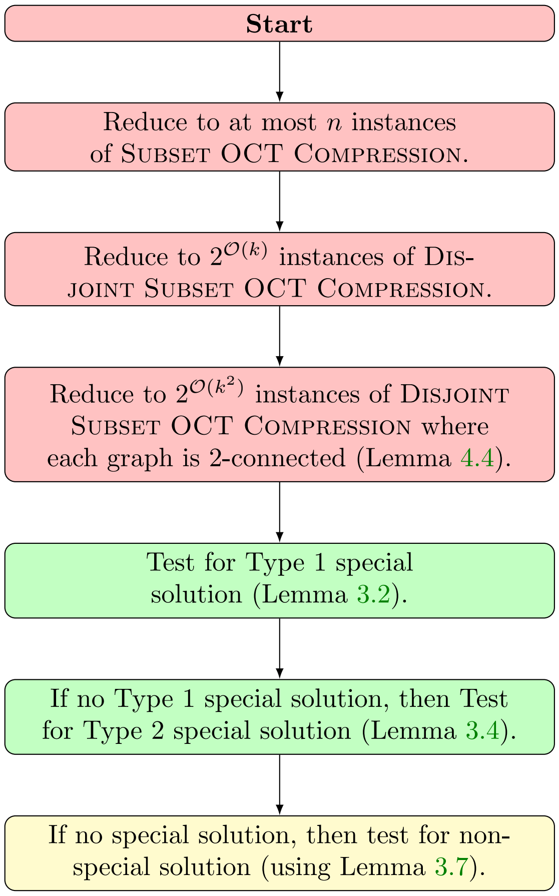
Start
Reduce to at most n instances of SUBSET OCT COMPRESSION.
Reduce to $2^{\mathcal{O}(k)}$ instances of DISJOINT SUBSET OCT COMPRESSION.
Reduce to $2^{\mathcal{O}(k^2)}$ instances of DISJOINT SUBSET OCT COMPRESSION where each graph is 2-connected (Lemma 4.4).
Test for Type 1 special solution (Lemma 3.2).
If no Type 1 special solution, then Test for Type 2 special solution (Lemma 3.4).
If no special solution, then test for non-special solution (using Lemma 3.7).

FIG. 2. *A summary of the main steps of our algorithm.*

Let S be a solution for this instance. Let $W_1 \uplus W_2$ be a partition of W such that W_1 occurs in a block in $G \setminus S$ and let $W_2 = W \setminus W_1$. We guess W_1 and W_2 by a $2^{\mathcal{O}(k)}$-way branching. Furthermore, suppose that there is a vertex p such that W_1 is semi-isolated from $W_2 \cup p$ in $G \setminus S$. If $p \in W_1$, then we simply guess p and now, since p is known, we run the algorithm of Lemma 3.5 and to account for the case when $p \notin W_1$, we also run the algorithm of Lemma 3.7. In either case, we compute a set of $2^{\mathcal{O}(k^2)}$ vertices which intersects a solution. We branch on the set thus computed and recurse on each instance obtained by deleting a vertex of this set from the instance and reducing the parameter k by 1.

The fact that the resulting search tree has a $2^{\mathcal{O}(k^2)}$-way branching with the budget k dropping by 1 in each branch, along with the fact that the time spent at each node is bounded by $2^{\mathcal{O}(k^3)} mn \log^2 n$, implies the bound on the running time. This completes the description of our algorithm for DISJOINT SUBSET OCT COMPRESSION. ∎

We conclude this section by summarizing all the steps in our algorithm for SUBSET OCT (Figure 2). We present the remaining details of our algorithm starting with the next section.

4. Reduction to 2-connected case.

This reduction is achieved in the following way. We first remove vertices which do not occur in a cycle along with a vertex of W, as well as bridges from the graph G. Then, every maximal 2-connected component (or block) of G contains a vertex of W and using the fact that the number of vertices in W is at most $k+1$ along with the fact that the block graph of G is a tree, we conclude that the number of cut vertices in G is bounded by $2k+2$. Following this, we branch into 2^{2k+2} cases by guessing the cut vertices of G which are part of some solution for the given instance and recurse whenever the guess is non-empty. When the guess is empty, we simply solve the problem independently on each block, which we show is equivalent to solving the problem on the given instance. We now proceed to the formal description of this reduction.

We begin by performing the following preprocessing step on the given instance (G, T, W, k) of Disjoint Subset OCT Compression.

Reduction Rule 4.1. *Given an instance (G, T, W, k) of* Disjoint Subset OCT Compression*, if there is a vertex v which does not lie on a T-cycle or a W-cycle, then return the instance $(G \setminus \{v\}, T, W, k)$. Similarly, if there is a bridge e in G, then return the instance $(G \setminus \{e\}, T, W, k)$.*

It is clear that the instance returned by an application of the above rule is equivalent to the original instance and the rule can be applied in time $\mathcal{O}(m+n)$ by simply computing the block decomposition of G using the algorithm of [12] and deleting every bridge of G and all vertices which occur in a block disjoint from T or disjoint from W. We now show that this reduction rule bounds the number of cut vertices in G. An instance on which the above reduction rule cannot be applied is called *irreducible*.

Lemma 4.2. *Let (G, T, W, k) be an instance of* Disjoint Subset OCT Compression *which is irreducible. Then, the number of cut vertices in G is at most $2k+2$.*

Proof. Since the reduction rule does not apply, it must be the case that every block contains a vertex of W. We also know that the block graph is a tree and the leaves of the tree are all vertices corresponding to blocks. We now root the block tree at an arbitrary cut vertex and also fix a total ordering of the vertices in W. Now, for every cut vertex v which is not in W, we charge v to the smallest vertex (in the total ordering) of W which appears in a block which is a child of this cut vertex. Since every block vertex in the block graph has a unique cut vertex as its parent, we will never charge the same vertex of W to 2 distinct cut vertices, which implies that the number of cut vertices disjoint from W is bounded by $k+1$. Therefore, we conclude that the number of cut vertices in G is at most $2k+2$. ∎

Lemma 4.3. *Let $I = (G, T, W, k)$ be an irreducible instance of* Disjoint Subset OCT Compression. *Let $B_1, \ldots, B_\ell$ be the blocks of G. Consider the graph G' obtained by taking the disjoint union of the graphs $B_1, \ldots, B_\ell$. Let T' be the set of copies of all the terminals (which now occur in different connected components) and let W' be the set of copies of all vertices of W. If I has a solution disjoint from the cut vertices of G then $I' = (G', T', W', k)$ is a* Yes *instance of* Disjoint Subset OCT Compression *and if I' is a* Yes *instance then I is a* Yes *instance as well.*

Proof. Let S be a solution for the instance I disjoint from the cut vertices of G. Then, the restriction of S to each block gives a partition of S among the blocks of G and since every cycle lies within a block of G, S is also a solution for I'. The converse direction of the statement is straightforward. ∎

We should note here that there is a slight abuse of definition in the statement of the previous lemma since the set W' could be much larger than W if some vertices of W occur in many blocks and therefore the instance (G', T', W', k) does not fit the definition of instances of the Disjoint Subset OCT Compression problem. However, we ignore this abuse since we will finally work on each connected component separately, which allows us to construct valid instances for each connected component.

By combining Lemma 4.2 and Lemma 4.3, we show how to reduce the given instance of Disjoint Subset OCT Compression to $2^{\mathcal{O}(k^2)}$ instances where the graph in each instance is 2-connected and the given input instance is a Yes instance if and only if at least one of the reduced instances is a Yes instance. More formally, we have the following lemma.

Lemma 4.4. *There is an algorithm that, when given an irreducible instance $I = (G, T, W, k)$ of* Disjoint Subset OCT Compression*, runs in time $2^{\mathcal{O}(k^2)}m$ and returns $\ell = 2^{\mathcal{O}(k^2)}$ instances $\{(G_i, T_i, W_i, k_i)\}_{1 \leq i \leq \ell}$ of* Disjoint Subset OCT Compression *such that every connected component of every G_i is 2-connected and furthermore, (G, T, W, k) is a* Yes *instance if and only if there is an $1 \leq i \leq \ell$ such that the instance (G_i, T_i, W_i, k_i) is a* Yes *instance.*

Proof. By Lemma 4.2, we know that G has at most $2k + 2$ cut vertices. Let Z be the set of cut vertices of G. For each $Z' \subseteq Z$, we construct an instance $I_{Z'} = (G \setminus Z', T \setminus Z', W \setminus Z', k - |Z'|)$ and recursively apply the same procedure on it. Clearly, I is a Yes instance if and only if there is a $Z' \subseteq Z$ such that $I_{Z'}$ is a Yes instance. For every non-empty choice of Z', we have reduced the parameter by at least 1. However, when $Z' = \emptyset$, we apply Lemma 4.3 to construct an instance $I' = (G', T', W', k)$ where every connected component is 2-connected. Therefore, this branching algorithm is a 2^{2k+2}-way branching with one branch being a leaf and the remaining branches achieving a reduction in the parameter by at least 1. Therefore, the algorithm returns $2^{\mathcal{O}(k^2)}$ instances such the original instance is a Yes instance if and only if one of these instances is a Yes instance and furthermore, every connected component of every returned instance is 2-connected. The running time of the algorithm follows from the fact that the time required to construct each returned instance is $\mathcal{O}(m)$. ∎

Since we can work on each connected component of a given instance independently, we will assume (at a multiplicative cost of a factor of k to the number of returned instances) that the graph in each instance returned by the algorithm of Lemma 4.4 is 2-connected. Furthermore, in order to avoid even more cumbersome problem names, we will henceforth only consider instances of Disjoint Subset OCT Compression where the graph is 2-connected and avoid framing yet another problem to denote the '2-connected' version of Disjoint Subset OCT Compression.

5. Isolated and Semi-isolated vertices.

In this section, we show that given a vertex which is isolated (Lemma 5.1) or semi-isolated (Lemma 5.16) from W in the graph obtained by removing a solution for the given instance, we can always compute a bounded set of vertices which intersects *some* solution for the same instance.

5.1. Computing a solution vertex using an isolated vertex.

Lemma 5.1. *Let (G, T, W, k) be an instance of* Disjoint Subset OCT Compression *and let S be a solution for this instance. Let v be a vertex isolated from W in $G \setminus S$. Given v, in time $\mathcal{O}(4^k km)$, we can find a set of at most $4^{k+1}(k+1)$ vertices which has a non-empty intersection with some solution for this instance.*

Proof. We first show that there is a solution for the given instance which intersects an important v-W separator of size at most $k+1$. Due to Preprocessing Rule 4.1, v lies on a W-cycle in G. Since v is isolated from W in $G \setminus S$, it must be case that there is a non-empty subset $K \subseteq S$ and a set $p(K)$ which is either empty or a singleton, such that $A = K \cup p(K)$ is a minimal v-W separator.

If A is an important v-W separator, then S itself is a solution of the claimed type. Suppose that A is not an important v-W separator and let B be an important v-W separator dominating A. We now select a vertex in B denoted by q as follows. If $p(K) = \emptyset$, then set $q = \emptyset$ and if $p(K) \in B$, then set $q = p(K)$ and if $p(K) \notin B$, then set as q an arbitrary vertex of $B \setminus A$ and let $J = B \setminus \{q\}$. Define S' as $S' = (S \setminus K) \cup J$. It is clear that S' is no larger than S. We claim that S' is also a solution for the instance.

If this were not the case, then there is an odd T-cycle C which intersects $S' \setminus S = K \setminus J$ in $G \setminus S'$. Since W is a T-oct for the given instance, C also intersects W in $G \setminus S'$. Since $K \setminus J$ is a subset of $R(v, B)$, any cycle containing a vertex of $K \setminus J$ and a vertex of W must intersect B in at least 2 vertices. Since all but at most 1 vertex of B are contained in S', the cycle C must intersect S', a contradiction. Therefore, we conclude that S' is a solution for the given instance which intersects an important v-W separator of size at most $k+1$.

Hence, the union of the vertices in the set of important v-W separators of size at most $k+1$, intersects some solution for the given instance. By Lemma 2.10 we know that the number of important v-W separators of size at most $k+1$ is at most 4^{k+1} and these can be enumerated in time $\mathcal{O}(4^k(m+n))$. Therefore, we simply enumerate all important v-W separators of size at most $k+1$ and return the vertices in the union of these separators. This completes the proof of the lemma. □

5.2. Computing a solution vertex using a semi-isolated vertex. In this subsection, the objective is to prove a lemma analogous to Lemma 5.1, but for semi-isolated vertices. However, the proof of this lemma, while similar in spirit to the previous one, is much more involved due to the fact that such vertices occur in the same block as some vertex of W when the solution is removed. In this case it is not necessary that some solution intersects a set of important separators. Hence, we require the idea of the generalization of important separators introduced in [20] to design an algorithm for semi-isolated vertices. We will prove a much more general lemma and show how one can plug in the appropriate values to obtain the required analogous lemma for semi-isolated vertices. We begin with the following structural lemma.

Lemma 5.2. *Let $G = (V, E)$ be a 2-connected graph and $T \subseteq V$ be a set of vertices. Then, G does not contain odd T-cycles if and only if G is bipartite.*

Proof. Suppose that G contains an odd cycle C. Let $t_1 \in T$ be a terminal. Clearly, C cannot intersect t_1. Since the graph is 2-connected, there are 2 vertex disjoint paths from t_1 to C. Let $c_1, c_2 \in C$ be such that there are vertex disjoint paths P_1 and P_2 from t_1 to c_1 and c_2 respectively such that these paths each intersect exactly one vertex of C. Let A_1 and A_2 be the two paths between c_1 and c_2 along C. Since C is an odd cycle, A_1 and A_2 are paths of opposite parity. Hence, depending on the parity of the paths P_1 and P_2, we can chose either A_1 or A_2 and construct an odd cycle passing through t_1, which is a contradiction. The other direction of the statement is obvious. This completes the proof of the lemma. □

The following observation is a consequence of Lemma 5.2.

Observation 5.3. *Consider an instance $I = (G, T, W, k)$ of* Disjoint Subset

OCT COMPRESSION *and a solution S for this instance. Let D be a block in $G \setminus S$. If $V(D)$ contains a terminal, then D is bipartite.*

DEFINITION 5.4. *Given a graph $G = (V, E)$ two vertices u and v, if there is a block in G containing both vertices, then we denote by $\mathbf{B}_G[u, v]$ this unique block. For a vertex set X with at least 2 vertices if there is a block in G containing X, then we denote this unique block by $\mathbf{B}_G[X]$ with the subscript dropped if it is clear from the context.*

We now introduce the notion of 'well domination'. This is one of the central notions used in the paper. For a pair of sets Q_1, Q_2, we use the notion of well domination to define (when possible) a relation between pairs of Q_1-Q_2 separators. We will state the definition first and then prove a lemma which will provide a clearer idea of the reason behind introducing this notion.

DEFINITION 5.5. *Let (G, T, W, k) be the given instance of* DISJOINT SUBSET OCT COMPRESSION. *Let Q_1 and Q_2 be disjoint vertex sets and piv a vertex. Let X be a vertex set disjoint from $Q_1 \cup Q_2 \cup \mathsf{piv}$ and $\ell \leq k$ such that,*

- *$|Q_1| \leq k$.*
- *(X, piv) is a minimal Q_1 - $(Q_2 \setminus \mathsf{piv})$ almost separator in G.*
- *The size of the smallest T-oct in the graph $G[R(Q_1, X \cup \mathsf{piv}) \cup \mathsf{piv}]$ is ℓ.*

Suppose that $(\hat{X}, \mathsf{piv})$ is a Q_1-$(Q_2 \cup \mathsf{piv})$ almost separator such that,

- *The separator $\hat{X} \cup \mathsf{piv}$ dominates the separator $X \cup \mathsf{piv}$ in G.*
- *There is a T-oct of size at most ℓ in the graph $G[R(Q_1, \hat{X} \cup \mathsf{piv}) \cup \mathsf{piv}]$.*

*Then, we say that $(\hat{X}, \mathsf{piv})$ **well dominates** (X, piv).*

The above definition is motivated by the following lemma.

LEMMA 5.6. *Let (G, T, W, k) be an instance of* DISJOINT SUBSET OCT COMPRESSION *and let S be a solution for this instance. Let Q_1, Q_2, piv and $X \subseteq S$ be such that they satisfy the conditions of Definition 5.5. Let X be inclusion-wise minimal subject to it satisfying these conditions. If $(\hat{X}, \mathsf{piv})$ is a Q_1-Q_2 almost separator which well dominates (X, piv), then there is a solution for the given instance which contains $\hat{X}$.*

Proof. Let K be a smallest T-oct for the sub-instance I induced on the graph $G[R(Q_1, X \cup \mathsf{piv}) \cup \mathsf{piv})]$ and let $\hat{K}$ be a smallest T-oct for the sub-instance I' induced on the graph $G[R(Q_1, \hat{X} \cup \mathsf{piv}) \cup \mathsf{piv})]$. We claim that the set $S' = (S \setminus (X \cup K)) \cup (\hat{X} \cup \hat{K})$ is also a solution for the given instance. If this were not the case then there is an odd T-cycle C in $G \setminus S'$. Observe that C must intersect $(X \cup K) \setminus (\hat{X} \cup \hat{K})$. Let $x \in (X \cup K) \setminus (\hat{X} \cup \hat{K})$ such a vertex in C.

Since $(\hat{X} \cup \mathsf{piv})$ dominates $(X \cup \mathsf{piv})$ in G, $x \in R_G(Q_1, \hat{X} \cup \mathsf{piv})$. Therefore, C cannot intersect a vertex in $NR(Q_1, \hat{X} \cup \mathsf{piv})$, which implies that C is an odd T-cycle in the graph $G[R(Q_1, \hat{X} \cup \mathsf{piv}) \cup \mathsf{piv})]$. However, by our assumption, $\hat{K}$ intersects every such odd T-cycle, and since $\hat{K} \subseteq S'$, we have a contradiction. This completes the proof of the lemma. ∎

5.2.1. Tight Separator Sequences. In this part of the paper, we present the definition of tight separator sequences, which provide a starting point for our generalization of important separators.

DEFINITION 5.7. *Let $G = (V, E)$ be a graph and let $X, Y \subseteq V$ be disjoint vertex sets. We define an important X-Y separator of order i, S^i to be the unique smallest important X-S^{i-1} separator in G, where $S^0 = Y$.*

Definition 5.8. *We define a tight X-Y separator sequence $\mathcal{I}$ to be a set $\mathcal{I} = \{S^i | 1 \leq i \leq l\}$, where S^i is an important X-Y separator of order i, for every $1 \leq i, j \leq l$, $|S^i| = |S^j|$, and $\lambda(X, S^l) > \lambda(X, Y)$, that is there is no X-S^l cut of size $|S^l|$.*

Observation 5.9. *Given two X-Y separators S_1 and S_2, we say that $S_1 \preceq S_2$ if S_2 covers S_1 with respect to X. Then, $(\mathcal{I}, \preceq)$ forms a total order where $\mathcal{I}$ is a tight X-Y separator sequence.*

By Lemma 2.9, for every i, an important X-Y separator of order i is unique and can be computed in polynomial time and therefore, a tight X-Y separator sequence is unique and can be computed in polynomial time. In fact, by using the algorithm of Lemma 2.4 from [25], we can compute the tight X-Y separator sequence in linear time.

Lemma 5.10. *There is an algorithm that, given a graph $G = (V, E)$, vertex sets X and Y, runs in time $\mathcal{O}(\ell(m + n))$ and returns the tight X-Y separator sequence where ℓ is the size of a minimum X-Y separator, $v = |V|$ and $m = |E|$.*

Lemma 5.11. *Let P_1 and P_2 be two separators in $\mathcal{I}$ such that $P_1 \prec P_2$ and there is no P_3 in $\mathcal{I}$ such that $P_1 \prec P_3 \prec P_2$. Then, the size of a minimum X-Y separator which lies in the set $NR(X, P_1) \cap R(X, P_2)$ is at least $|P_1| + 1$.*

We now move on to the main lemma of this subsection which, as we will argue immediately after the proof, implies an algorithm to compute a solution vertex starting from a semi-isolated vertex. Essentially, given a vertex v, which is semi-isolated from W in $G \setminus S$ for some solution S with the pivot being $w \in W$, we will (in the following lemma) set $Q_1 = \{v\}$, $Q_2 = W$, $\mathsf{piv} = w$. The output of the algorithm of this lemma will contain either a solution vertex or an isolated vertex, from which we already know how to compute a solution vertex.

Lemma 5.12. *Let (G, T, W, k) be an instance of* Disjoint Subset OCT Compression, *S be a solution for this instance, let Q_1 and Q_2 be disjoint vertex sets and let piv denote a vertex such that*

- *$|Q_1| \leq k$*
- *Q_1 is semi-isolated from $Q_2 \cup \mathsf{piv}$ with piv being the pivot between Q_1 and $Q_2 \cup \mathsf{piv}$ in $G \setminus S$. Furthermore, the vertices of Q_1 occur in the same block of $G \setminus S$, which we denote by $\mathbf{B}[Q_1]$.*
- *there is a set $X \subseteq S$ of size at most k such that (X, piv) is a minimal Q_1-Q_2 almost separator.*

Let ℓ be the size of the smallest T-oct for the graph $G[R(Q_1, X \cup \mathsf{piv}) \cup \mathsf{piv}]$ and let $\mathcal{A}$ be an algorithm that, on input Q_1, Q_2 and an almost Q_1-Q_2 separator (X', piv), runs in time $T(\ell, m, n)$ and decides whether there is a T-oct of size at most ℓ for the graph $G[R(Q_1, X' \cup \mathsf{piv}) \cup \mathsf{piv}]$. Then, there is an algorithm that on input $G, T, W, k, Q_1, Q_2, \mathcal{A}$ and piv, runs in time $2^{\mathcal{O}(k^2)} T(\ell, m, n) \log n$ and returns a set $\mathcal{R}$ of $2^{\mathcal{O}(k^2)}$ vertices such that

- *$\mathcal{R}$ intersects some solution for the given instance or*
- *$\mathcal{R}$ contains a vertex which is isolated or semi-isolated from $Q_1 \cup Q_2 \cup \mathsf{piv}$ in $G \setminus S'$ where S' is a solution (possibly distinct from S). In this case, it also contains a vertex which is isolated from $Q_2 \cup \mathsf{piv}$ in $G \setminus S'$*

Proof. We first perform the following edit on the given instance to make future steps of the algorithm easier. We first guess whether the block $\mathbf{B}[Q_1]$ in the graph $G \setminus S$ contains a terminal or not. In the case where we assume that this block does not contain a terminal, we make a clique on $Q_1 \cup \mathsf{piv}$. In the case when we assume that this block contains a terminal, Lemma 5.2 implies that this block is bipartite. Then, we guess a bipartition of $Q_1 \cup \mathsf{piv}$ which is "consistent" with a bipartition of the block $\mathbf{B}[Q_1]$ in $G \setminus S$. After this, we add edges (if it is not already there) between every pair of vertices guessed to be in different partitions and add subdivided edges between every pair of vertices guessed to be in the same partition. This edit is done simply to ensure that henceforth, there is always a block containing $Q_1 \cup \mathsf{piv}$ in any recursive step of the algorithm (conserving the parities of the paths between them appropriately). Furthermore, since the size of Q_1 is at most k, we can perform this edit in time $\mathcal{O}(k^2(m+n))$ and we will add $\mathcal{O}(k^2)$ new vertices to the graph. Note that the vertices added to the graph during the subdivision will be considered undeletable. The notion of vertices being undeletable will be formalized later.

Note that the algorithm we describe depends on the structure of the block $\mathbf{B}[Q_1]$ in $G \setminus X$, resulting in two cases (based on whether this block contains a terminal or not). While executing the algorithm, we simply execute the steps of the algorithm corresponding to both cases.

Objective of the algorithm.. Formally, given $I = (G, Q_1, \mathsf{piv}, Q_2, Z, T, k)$, where $Z \subseteq V$ is disjoint from $(Q_1 \cup Q_2 \cup \mathsf{piv})$, the algorithm returns a set $\mathcal{R} \subseteq Z$ of vertices such that for any Q_1-Q_2 almost separator (X, piv) in the graph G

- Either $\mathcal{R}$ intersects an almost separator $(\hat{X}, \mathsf{piv})$ well dominating (X, piv).
- Or there is a vertex in $\mathcal{R}$ which is isolated or semi-isolated from $Q_1 \cup Q_2 \cup \mathsf{piv}$ in $G \setminus (\hat{X} \cup \hat{K})$ where $(\hat{X}, \mathsf{piv})$ is a Q_1-Q_2 almost separator well dominating (X, piv). Further there is a vertex in $\mathcal{R}$ which is isolated from $Q_2 \cup \mathsf{piv}$ in $G \setminus (\hat{X} \cup \hat{K})$.

Therefore, if $\mathcal{R}$ satisfies the first condition above, then by Claim 5.6 there is a solution S' containing $\hat{X}$ and so $\mathcal{R}$ intersects some solution S'. Or else $\mathcal{R}$ satisfies the second condition. So there is a vertex in $\mathcal{R}$ which is isolated or semi-isolated from $Q_1 \cup Q_2 \cup \mathsf{piv}$ in $G \setminus S'$. Furthermore, there is a vertex in $\mathcal{R}$ which is isolated from $Q_2 \cup \mathsf{piv}$ in $G \setminus S'$. Thus the set $\mathcal{R}$ satisfies all the conditions required by the lemma.

In the above, Z denotes the set of *deletable vertices* in the instance I. As we shall see shortly, in the recursive calls to our algorithm, Z will be narrowed down further. For the sake of a cleaner presentation, for the rest of the proof of this lemma, we adopt the convention that whenever we refer to a set being an s-t separator without explicitly mentioning a graph, we mean that the set is a s-t separator in the graph $G \setminus \mathsf{piv}$ and whenever we refer to a pair being an almost s-t separator, we are referring to separation in the graph G.

Part I: The algorithm.. We first check whether there is a Q_1-Q_2 separator of size at most k contained in the given subset Z. If not, then we conclude that the input is invalid (the bound on the size of X in the premise is violated). If there is no path from Q_1 to Q_2 in $G \setminus \mathsf{piv}$, then we return $\emptyset$. Otherwise, we compute the tight Q_1-Q_2 separator sequence in the graph $G \setminus \mathsf{piv}$, comprising only the vertices in Z. This takes time $\mathcal{O}(k(m+n))$ by Lemma 5.10. Let $K = S \cap R(Q_1, X \cup \mathsf{piv})$ and $|K| = \ell$. We iterate over all choices of ℓ in $[1, k]$.

We call a Q_1-Q_2 separator P *good* if there is a T-oct of size at most ℓ for the instance induced on the graph $G[R(Q_1, P \cup \mathsf{piv}) \cup \mathsf{piv}]$ and we call it *bad* otherwise. The following observation plays a crucial role in allowing us to ignore (potentially) large parts of the graph during our search.

Observation 5.13. *If a Q_1-Q_2 separator is* good*, all Q_1-Q_2 separators covered*

```
Input  : $(G, Q_1, \mathsf{piv}, Q_2, Z, T, k)$
Output: A subset of $Z$ which, for every $Q_1$-$Q_2$ almost separator $(X, \mathsf{piv})$,
        either intersects a $Q_1$-$Q_2$ almost separator $(X', \mathsf{piv})$ well dominating
        $(X, \mathsf{piv})$ or contains a vertex isolated or semi-isolated from
        $Q_1 \cup Q_2 \cup \mathsf{piv}$ in $G \setminus X$ and No if no such $Q_1$-$Q_2$ almost separator
        exists
1  if $k < 0$ then return No
2  Check if there is a $Q_1$-$Q_2$ separator of size at most $k$ in the graph $G \setminus \mathsf{piv}$
   which is contained in $Z$
3  if there is no such separator then return No
4  if there is no $Q_1$-$Q_2$ path in $G \setminus \mathsf{piv}$ then return $\emptyset$
5  $\mathcal{I}$ =TIGHT-SEPARATOR-SEQUENCE$(G \setminus \mathsf{piv}, Q_1, Q_2, k)$
6  for each $\ell < k$ do
7      $P_1 \leftarrow$ maximal good separator (depends on $\ell$)
8      $P_2 \leftarrow$ minimal bad separator (depends on $\ell$)
9      $\mathcal{R} \leftarrow (P_1) \cup (P_2 \setminus Q_2)$
10     $Z' = Z \cap (R_{G\setminus\mathsf{piv}}(Q_1, P_2) \cap NR_{G\setminus\mathsf{piv}}(Q_1, P_1))$
11     $\mathcal{R} \leftarrow \mathcal{R} \cup$ SEMI-ISO$(G, Q_1, \mathsf{piv}, Q_2, Z', T, k)$
12     for each ordered 3-partition $J = (P_1^{nr}, P_1^e, P_1^o)$ of $P_1$ do
13         $\mathcal{R} \leftarrow \mathcal{R} \bigcup \cup_{k' \leq k-1}$ SEMI-ISO$(G_J, Q_1, \mathsf{piv}, P_1^{nr}, Z \cap G_J, T \cup (P \setminus P_1^{nr}), k')$
14         $\mathcal{R} \leftarrow \mathcal{R} \bigcup \cup_{k' \leq k-1}$ SEMI-ISO$(G_J, Q_1, \mathsf{piv}, P_1^{nr}, Z \cap G_J, T, k')$
15     end
16     for each ordered 3-partition $J = (P_2^{nr}, P_2^e, P_2^o)$ of $P_2$ do
17         $\mathcal{R} \leftarrow \mathcal{R} \bigcup \cup_{k' \leq k-1}$ SEMI-ISO$(G_J, Q_1, \mathsf{piv}, P_2^{nr}, Z \cap G_J, T \cup (P \setminus P_2^{nr}), k')$
18         $\mathcal{R} \leftarrow \mathcal{R} \bigcup \cup_{k' \leq k-1}$ SEMI-ISO$(G_J, Q_1, \mathsf{piv}, P_2^{nr}, Z \cap G_J, T, k')$
19     end
20 end
21 return $R$
```

Algorithm 5.1: Algorithm SEMI-ISO

by this separator are also good *and if a Q_1-Q_2 separator is* bad*, all Q_1-Q_2 separators which cover this separator are* bad*.*

We now compute the maximal element of $\mathcal{I}$ which is good. We can do this by solving Subset OCT on the sub-instances corresponding to each separator in $\mathcal{I}$. Each of these tests takes time $T(\ell, m, n)$. Since we need to run this test for each separator in $\mathcal{I}$, this takes time $T(\ell, m, n)n$.[1]

Let P_1 be the maximal element of $\mathcal{I}$ which is good and let P_2 be the minimal element of $\mathcal{I}$, which is bad. That is, P_1 is good and every separator in $\mathcal{I} \setminus \{P_1\}$ which covers P_1 is bad, P_2 is bad and every separator in $\mathcal{I} \setminus \{P_2\}$ covered by P_2 is good. If all the separators in $\mathcal{I}$ are good, then P_2 is set as Q_2 and if all separators in $\mathcal{I}$ are bad, then P_1 is set as Q_1. We now describe the rest of the algorithm. This is broken up into 4 cases depending on the interaction between X, P_1 and P_2 in G.

[1] However, in order to find the maximal element of $\mathcal{I}$ which is good, we perform these tests in the form of a binary search over the separators in $\mathcal{I}$ which are by definition in a "sorted" form. Though we do not give the details of implementation of this step, it should be fairly obvious that finding the maximal good separator of $\mathcal{I}$ can be done in time $T(\ell, m, n) \log n$.

1. P_1 well dominates the separator X or $(P_1 \cup P_2) \cap X \neq \emptyset$.. We add the vertices in P_1 and $P_2 \setminus Q_2$ into the set $\mathcal{R}$.

2. The separator X covers P_1, and X is covered by P_2.. We set $Z' = Z \cap (R_{G\setminus \mathsf{piv}}(Q_1, P_2) \cap NR_{G\setminus \mathsf{piv}}(Q_1, P_1))$ and then add to $\mathcal{R}$ the vertices returned by the recursive call SEMI-ISO $(G, Q_1, \mathsf{piv}, Q_2, Z', T, k)$ (see Algorithm 5.1). This step is intended for the case when the set X covers P_1 but X is in turn covered by P_2 in $G \setminus \mathsf{piv}$.

3. X and P_1 are incomparable in $G \setminus \mathsf{piv}$.. We consider all ordered 3-partitions of P_1, $J = (P_1^{nr}, P_1^e, P_1^o)$ where the set P_1^{nr} denotes the guess of those vertices of P_1 which are *not reachable* from Q_1 in the graph $G \setminus (X \cup \mathsf{piv})$, the set $P_1 \setminus P_1^{nr}$ denotes the guess of those vertices of P_1 which are reachable from Q_1 in $G \setminus (X \cup \mathsf{piv})$. We also assume that the vertices in P_1^r lie in the same block of $G \setminus S$ as Q_1. The correctness behind this assumption will be argued later. Now, for each such 3-partition $J = (P_1^{nr}, P_1^e, P_1^o)$, we construct a graph G_J as follows. Initially, we set $G_J = G[R(Q_1, P_1) \cup P_1 \cup \mathsf{piv}]$. Let $P_1^r = P_1 \setminus P_1^{nr}$. We now have one of the following two cases,

(a) The block $\mathbf{B}[Q_1]$ in $G \setminus S$ is bipartite. Then for every pair in $P_1^e \times P_1^o$, we add an edge between the vertices and for every pair in $\binom{P_1^e}{2}$ and $\binom{P_1^o}{2}$ we add a subdivided edge between the vertices. This completes the construction of G_J.

(b) The block $\mathbf{B}[Q_1]$ in $G \setminus S$ is T-cycle free. Then we add an edge and a subdivided edge (if one does not already exist) between every pair of vertices in $Q_1 \cup (P_1 \setminus P_1^{nr})$. This completes the construction of G_J. Now, for each G_J, for each $1 \leq k' \leq k-1$, we recurse on the instance $(G_J, Q_1, \mathsf{piv}, P_1^{nr}, Z \cap G_J, T, k')$ and add the vertices in the sets returned, to $\mathcal{R}$.

4. X and P_2 are incomparable.. We do the same for P_2 as we did for P_1.

Finally we return $\mathcal{R}$.

This completes the description of the algorithm. We now prove the correctness of the algorithm.

Part II: Correctness.. For each tuple $I = (G, Q_1, \mathsf{piv}, Q_2, Z, T, k)$ on which the algorithm is invoked, we define a measure $\mu(I) = 2k - \lambda$ where λ is the size of the smallest Q_1-Q_2 separator in the graph $G \setminus \mathsf{piv}$ which is completely contained in Z. We prove the correctness of the algorithm by induction on the measure $\mu(I)$.

In the base case, if $\lambda > k$, then the algorithm returns NO, which is clearly correct. Similarly, if $\lambda = 0$, then there is no path between Q_1 and W_2 in the graph $G \setminus \mathsf{piv}$ and hence the algorithm simply returns $\emptyset$ as the separator, which is also correct. This completes the correctness of the base cases and we now assume that the algorithm is correct on all instances I with $\mu(I) < \mu$. Now, consider an instance I such that $\mu(I) = \mu$.

1. P_1 well dominates the separator X or $(P_1 \cup P_2) \cap X \neq \emptyset$.. We add $P_1 \cup (P_2 \setminus Q_2)$ to $\mathcal{R}$. In the first case, by Claim 5.6 there is a solution containing P_1 and $\mathcal{R}$ contains it. In the second case, the set $\mathcal{R}$ intersects X by our assumption. From now on, we shall assume that X is disjoint from $P_1 \cup P_2$, as there is no reason to proceed otherwise.

2. The separator X covers P_1, and X is covered by P_2.. In this case we restrict Z to the subset Z', which contains only those vertices which lie between the separators P_1 and P_2. Let I' be the instance on which the algorithm is invoked recursively. By Lemma 5.11, any Q_1-Q_2 separator which lies in the set $NR_{G\setminus\mathsf{piv}}(Q_1, P_1) \cap R_{G\setminus\mathsf{piv}}(Q_1, P_2)$ has size at least $|P_1| + 1 = \lambda + 1$. Hence, $\mu(I') \leq \mu - 1$. Let $\mathcal{R}'$ be the set returned by the recursive call on I' which we add to $\mathcal{R}$. Then $\mathcal{R}$ satisfies all the required conditions, in this case.

3. The separator X is incomparable with P_1.. Let P_1^r be the intersection of P_1 and $R(Q_1, (X \cup \mathsf{piv}))$ and $P_1^{nr} = P_1 \setminus P_1^r$. Also, let X^r be the intersection of X with $R_{G\setminus\mathsf{piv}}(Q_1, P_1)$ and let X^{nr} be the rest of X. Since X is incomparable with P_1, by Observation 2.7, P_1^r, X^r, P_1^{nr} and X^{nr} are non empty. Recall that $K = S \cap R(Q_1, X \cup \mathsf{piv})$ and let K^r be the vertices of K in $R(Q_1, (P_1 \cup \mathsf{piv}))$ and $K^{nr} = K \setminus K^r$.

Suppose that there is some vertex $u \in P_1^r$ which is not in the block $\mathbf{B}[Q_1]$ of $G \setminus S$. Equivalently, u does not lie in the block $\mathbf{B}[Q_1]$ of the graph $G' = G \setminus (X \cup K)$. Then there is some cut vertex q which separates u and $Q_1 \cup \mathsf{piv}$. Note that piv and Q_1 always lie in the block $\mathbf{B}[Q_1]$ of G' because of the edit we made to the graph at the beginning. Since every path from u to Q_2 must intersect piv, it must also intersect q. Hence u is either isolated or semi-isolated from $Q_1 \cup Q_2 \cup \mathsf{piv}$ in G' (and in $G \setminus S$). Furthermore, recall that u remains connected to Q_1 in $G \setminus (X \cup \mathsf{piv})$. Therefore q is different from piv. Hence u is also isolated from $Q_2 \cup \mathsf{piv}$ in G' (and in $G \setminus S$).

Recall that we have already added the vertices of P_1 to $\mathcal{R}$. Hence if there were such an $u \in P_1^r$, $\mathcal{R}$ satisfies all the required conditions. So we shall assume that, P_1^r lies in the block $\mathbf{B}[Q_1]$. We now have two cases depending on the presence or absence of a terminal vertex in $\mathbf{B}[Q_1]$.

Case 1: *The block $\mathbf{B}[Q_1]$ contains a terminal in $G \setminus X$.* Therefore this block must be bipartite. Let P_1^e and P_1^o be the partition of P_1^r induced by some fixed bipartition of $\mathbf{B}[Q_1]$ and consider the graph G_J corresponding to the partition $J = (P_1^{nr}, P_1^e, P_1^o)$. Recall that an edge (x, y) (or a subdivided edge between x and y) in G_J where $x, y \in P_1^r$ is either present in G or corresponds to an x-y odd path (respectively even path) in the block $\mathbf{B}[Q_1]$ of $G \setminus S$. Any edge (subdivided edge) which is present in G_J but not in G is called a *torso* edge (subdivided edge).

Case 2: *The block $\mathbf{B}[Q_1]$ does not contain a terminal in $G \setminus X$.* Therefore it also does not contain any T-cycles. Let G_J be the graph corresponding to the partition $J = (P_1^{nr}, P_{11}, P_{12})$ where (P_{11}, P_{12}) is an arbitrary partition of P_1^r. Recall that in this case, an edge (x, y) in G_J where $x, y \in P_1^r$ is either present in G or corresponds to an x-y path in $G \setminus S$.

Let I' be the instance constructed in this step. Recall that the set returned by the invocation $\mathsf{SEMI\text{-}ISO}(I)$ contains the vertices in the set returned by the invocation $\mathsf{SEMI\text{-}ISO}(I')$. We now prove the following claim which will be used along with the induction hypothesis to prove the correctness of the algorithm on I.

Claim 5.14. **(a)** *If the block $\mathbf{B}[Q_1]$ in $G \setminus X$ does not contain T-cycles (is bipartite), then $\mathbf{B}[Q_1]$ in $G_J \setminus (X^r \cup K^r)$ does not contain T-cycles (respectively, is bipartite).*

(b) *Let Y^r be a v-P_1^{nr} separator in G_J such that (Y^r, piv) well dominates (X^r, piv) in G_J. Then, the set $X' = (X \setminus X^r) \cup Y^r$ is such that (X', piv) is a Q_1-Q_2 almost separator well dominating (X, piv) in G.*

(c) *Let Y^r be a Q_1-P_1^{nr} separator in G_J such that (Y^r, piv) well dominates (X^r, piv)*

in G_J. Let $X' = (X \setminus X^r) \cup Y^r$. If a vertex u is isolated from or semi-isolated from $Q_1 \cup P_1^{nr} \cup \mathsf{piv}$ in $G_J \setminus (Y^r \cup K_Y^r)$ where K_Y^r is a subset oct for the instance $G_J \setminus Y^r$, then u is isolated or semi-isolated from $Q_1 \cup Q_2 \cup \mathsf{piv}$ in $G \setminus (X' \cup K')$ where $K' = K_Y^r \cup K^{nr}$. Similarly, if u is isolated from $P_1^{nr} \cup \mathsf{piv}$ in $G_J \setminus (Y^r \cup K_Y^r)$, then u is isolated from $Q_2 \cup \mathsf{piv}$ in $G \setminus (X' \cup K')$.

Proof. **(a)** We first consider the case when $\mathbf{B}[Q_1]$ in $G \setminus S$ does not contain T-cycles. Suppose that $\mathbf{B}[Q_1]$ in $G_J \setminus (X^r \cup K^r)$ contains a T-cycle C. We replace every torso edge in C with the corresponding path between two vertices in P_1^r, which we have already concluded lie in the block $\mathbf{B}[Q_1]$ in $G \setminus S$. This operation results in a 2-connected subgraph of $G \setminus S$ which contains Q_1, piv, and a vertex in T, a contradiction.

We now consider the case when $\mathbf{B}[Q_1]$ in $G \setminus S$ is bipartite. Suppose that $\mathbf{B}[Q_1]$ in $G_J \setminus X^r$ contains an odd cycle C. If C does not contain torso edges, then C is an odd T-cycle in the block $\mathbf{B}[Q_1]$ in $G \setminus X$, a contradiction. Hence, C must contain torso edges. We replace each torso edge (or subdivided edge) by the corresponding paths of the same parity which we know exist in $G \setminus X$, to obtain a closed odd walk C' in $G \setminus X$. Since the vertices of P_1^r are in $\mathbf{B}[Q_1]$ in $G \setminus X$, C' is actually a closed odd walk in the block $\mathbf{B}[Q_1]$ in $G \setminus X$. Hence, C' contains an odd cycle which also lies in $\mathbf{B}[Q_1]$ in $G \setminus X$, a contradiction.

(b) Let K_Y^r be a subset oct for the sub-instance I' induced on G_J such that $|K_Y^r| \leq K^r$. Suppose that (X', piv) does not well dominate (X, piv) in G. This implies that the set $K' = K_Y^r \cup K^{nr}$ is not a subset oct for the sub-instance induced on $G' = G[R(Q_1, X' \cup \mathsf{piv}) \cup \mathsf{piv}]$. Consider an odd T-cycle in G' disjoint from K'. Clearly, this cycle must contain vertices of P_1^r. If there is a subpath of this cycle between vertices of P_1^r with the internal vertices disjoint from the vertices of G_J, then we can replace this path with a corresponding torso edge or sub-divided edge to get either a odd T-cycle in the block B_{Q_1} in $G_J \setminus (Y^r \cup K_Y^r)$ or an odd $(T \cup P_1^{nr})$-cycle in the block B_{Q_1} in $G_J \setminus (Y^r \cup K_Y^r)$, which is a contradiction.

(c) The first part follows from the fact that if u has 2 vertex disjoint paths to $Q_1 \cup Q_2 \cup \mathsf{piv}$ in the graph $G \setminus (X' \cup K')$, then we can replace the subpaths of these paths which are not present in G_J with the corresponding torso edges or subdivided edges (which are also clearly vertex disjoint) to get 1 vertex disjoint paths from u to $Q_1 \cup P_1^{nr} \cup \mathsf{piv}$ in $G_J \setminus (Y^r \cup K_Y^r)$. We can similarly prove the second part. This completes the proof of the claim. ◻

Now, let Y^r be a Q_1-P_1^{nr} separator in G_J such that (Y^r, piv) well dominates (X^r, piv) in G_J. Then, due to Claim 5.14, $Y^r \cup X^{nr}$ is a Q_1-Q_2 separator which well dominates X in the graph G. Therefore, a set intersecting a Q_1-P_1^{nr} separator which well dominates X^r in the graph G_J also intersects a Q_1-Q_2 separator which well dominates X in the graph G. Furthermore, due to Claim 5.14, a vertex which is isolated or semi-isolated from piv in $G_J \setminus Y^r$ is also isolated or semi-isolated from $Q_1 \cup Q_2 \cup \mathsf{piv}$ in $G \setminus ((X \setminus X^r) \cup Y^r)$. Therefore, if the algorithm is correct on I', then it is also correct on I and hence it only remains to prove that the algorithm is correct on I'. In order to prove that the algorithm is correct on I', it is sufficient to prove that $\mu(I') < \mu(I) = \mu$ since the correctness then follows from the induction hypothesis.

By Menger's theorem, since P_1 is a minimum size Q_1-Q_2 separator, we know that there are P_1^{nr} vertex disjoint paths from Q_1 to P_1^{nr}, which is also a lower bound on the size of the smallest Q_1-P_1^{nr} separator in G_J. Now, $\mu(I) = 2(|X^r| + |X^{nr}|) - (|P_1^{nr}| + |P_1^r|)$ and $\mu(I') = 2|X^r| - |P_1^{nr}|$, which implies that $\mu(I') = \mu(I) - (2|X^{nr}| - |P_1^r|)$. Since $|X^{nr}| \geq |P_1^r|$, we have that $\mu(I) - \mu(I') \geq |X^{nr}|$. Since X^{nr} is non empty, we conclude that $\mu(I') < \mu(I)$, which completes the proof of correctness of this case.

4. The separator X is incomparable with P_2. This correctness of this case is analogous to the correctness of the previous case.

We note that the separator X is a good separator by definition and therefore cannot cover P_2 due to Observation 5.13. Therefore the case that X covers P_2 need not be taken into consideration and the cases we have considered are exhaustive. This completes the proof of correctness of the algorithm.

Part III: Running time analysis. We show by induction on $\mu(I)$ that the number of vertices in the set returned by SEMI-ISO(I), denoted by $N(\mu(I))$, is bounded by $2^{6\mu(I)^2}$. In each of the base cases of the algorithm, we either return a single vertex or say NO, and hence the bound clearly holds. We assume that the claimed bound is true for all instances with $\mu(I) < \mu$. Now, consider an instance I such that $\mu(I) = \mu$. Moreover, $k > 1$.

1. The number of vertices added in Line 8 is bounded by $2k$.

2. The number of vertices added in Line 10 is bounded by $2^{6(\mu-1)^2}$ by the induction hypothesis.

3. Consider the vertices added in Line 12. There are at most 3^k ordered 3-partitions of P_1 and k choices for k'. For each of these choices, we return a set of size at most $N(\mu - 1)$, which, by the induction hypothesis is at most $2^{6(\mu-1)^2}$. Hence, the number of vertices added in this step is bounded by $3^k \cdot k \cdot 2^{6(\mu-1)^2}$.

4. Similarly, the number of vertices added in Line 15 is bounded by $3^k \cdot k \cdot 2^{6(\mu-1)^2}$.

Using the fact that $k \leq \mu$ and $k \geq 1$, we note that the total number of vertices returned by SEMI-ISO(I) is at most $2^{6(\mu)^2}$, which yields the required bound. The number of leaves of the recursion tree is clearly bounded by the size of the set returned by the algorithm, which is $2^{\mathcal{O}(k^2)}$, which is also a bound on the number of internal nodes of the recursion tree. Since the algorithm spends $2^{\mathcal{O}(k)}(m+n) + \mathcal{O}(m \log n) + \mathcal{O}(k(m+n))$ time at each node of the search tree, the total time taken by the algorithm on the given input is $2^{\mathcal{O}(k^2)} m \log n$. Since we invoke the Algorithm SEMI-ISO for every $1 \leq k' \leq k$, the total number of vertices returned is $2^{\mathcal{O}(k^2)}$. This completes the proof of the lemma. ∎

We now use a special case of the above lemma to show that given a vertex semi-isolated from W, we can compute a set of $2^{\mathcal{O}(k^2)}$ vertices which intersects some solution. We begin with the following observation.

OBSERVATION 5.15. *Let (G, T, W, k) be an instance of* DISJOINT SUBSET OCT COMPRESSION *and let S be a solution for this instance. Let v be a vertex semi-isolated from W in $G \setminus S$, $w = \mathsf{piv}(v, W)$ and let $W_2 = W \setminus w$. Let X be a minimal part of S separating v from W_2 in the graph $G \setminus w$. Then, the graph $G[R(v, X \cup w) \cup w]$ does not contain odd T-cycles.*

Proof. Suppose that there is an odd T-cycle C in the graph $G' = G[R(v, X \cup w) \cup w]$. Let $X' \subseteq S \setminus X$ be the set of vertices of S in $R(v, X \cup w)$. By our assumption, X' is non-empty and X' is separated from W_2 by $X \cup w$. We claim that $S' = (S \setminus X') \cup w$ is also a T-oct of size at most k for the given instance. Since X' is non-empty, we have that $|S'| \leq k$. If S' is not a T-oct, then there is an odd T-cycle intersecting a vertex in X' and a vertex in W_2 (since W is a T-oct) which is not possible since $X \cup w$ is an X'-W_2 separator in G. Therefore, S' is a solution for the given instance which intersects W, contradicting our assumption that all input instances have the property that there is no T-oct of size at most k which intersects W. ∎

The above observation implies that if we take Q_1 as a vertex semi-isolated from W with respect to a solution S, $w = \mathsf{piv}_{G \setminus S}(v, W)$ and $Q_2 = W \setminus w$ then the size of the smallest T-oct in the graph $G[R(v, X \cup w) \cup w]$ is 0, where $X \subseteq S$ is a minimal v-$W \setminus w$ separator in $G \setminus w$. Therefore, algorithm $\mathcal{A}$ only has to test if there is an odd T-cycle in a particular subgraph and this can clearly be done in time $\mathcal{O}(m+n)$. This leads to the following special case of Lemma 5.12.

LEMMA 5.16. *Let (G, T, W, k) be an instance of* DISJOINT SUBSET OCT COMPRESSION, *S be a solution for this instance and let v be a vertex semi-isolated from W in $G \setminus S$ such that v lies in the same block as $w \in W$. There is an algorithm running in time $2^{\mathcal{O}(k^2)} m \log n$ which returns a set of $2^{\mathcal{O}(k^2)}$ vertices such that this set either intersects some solution for the given instance or contains a vertex which is isolated from W after the removal of some solution for the given instance.*

By combining this lemma with Lemma 5.1, we have the following lemma on computing a solution vertex when we are given a semi-isolated vertex.

LEMMA 5.17. *Let $I = (G, T, W, k)$ be an instance of* DISJOINT SUBSET OCT COMPRESSION, *S be a solution for this instance and let v be a vertex semi-isolated from W in $G \setminus S$. There is an algorithm that, given I and v, runs in time $2^{\mathcal{O}(k^2)} m \log n$ and returns a set of $2^{\mathcal{O}(k^2)}$ vertices such that this set intersects some solution for the given instance.*

6. Solving special instances of type 1.

This section is devoted to our algorithm for DISJOINT SUBSET OCT COMPRESSION in the case when we are interested in finding a special solution of Type 1. We restate this lemma for the sake of completeness.

LEMMA 6.1. *There is an algorithm that, given an instance (G, T, W, k) of* DISJOINT SUBSET OCT COMPRESSION, *runs in time $2^{\mathcal{O}(k^3)} m \log n$ and either returns a solution for the given instance or correctly concludes that no Type 1 solution exists.*

Proof. We first perform the following operation on the vertices of W which we show does not affect the fact that S is a special solution of type 1, and plays a part in simplifying future arguments. We make W a clique to obtain another instance I'.

CLAIM 6.2. *S is also a special solution of type 1 for I' and any type 1 solution for I' is a solution for I.*

Proof. It is clear that S is also a special solution of type 1 for I' since adding the clique edges between the vertices of W does not create new T-odd cycles disjoint from S. Similarly, any type 1 solution for I' is also a solution for I since removing edges from I' cannot create new T-odd cycles. ∎

Given the above claim, it is enough for us to compute a solution for the instance I'. For the sake of notational convenience, we will refer to the instance I' as I. Consider the set of terminals T. If this set has at most k terminals, then the set T itself is a solution for the given instance. Therefore, we may assume that $|T| > k+1$. In this case, for any set T' of $k+1$ terminals, at least one of the terminals in T' is in the isolated or semi-isolated part of $G \setminus S$. Hence, we simply guess a terminal $t_1 \in T'$ which lies in the semi-isolated or isolated part of $G \setminus S$ and then apply Lemma 5.1 and Lemma 5.17 on t_1.

The fact that $G[W]$ is a clique and the definition of "domination" and "well domination" in Lemma 5.1 and Lemma 5.16 together imply that the resulting instance is also a special instance of type 1, allowing us to recurse on this instance.

The correctness of this algorithm follows from the correctness of Lemma 5.1, Lemma 5.17 and the fact that the branching is exhaustive. The bound on the running time follows from that of Lemma 5.1 and Lemma 5.17 and the bound on the sets returned by their respective algorithms. ∎

7. Solving special instances of type 2.

In this section, we design an algorithm to solve special instances of Type 2.

7.1. Generalized important blocks.

Definition 7.1. *Given a connected graph $G = (V, E)$, and a set $\mathcal{B} = \mathbf{B}_1, \ldots, \mathbf{B}_\ell$ of blocks of G, let $b_1, \ldots, b_\ell$ be the vertices in the block tree T which correspond to these blocks. Furthermore, assume that the block tree is rooted at a particular block. We now remove all vertices from T except those that participate in a b_i-b_j path in the tree T. Consider the tree T' obtained by repeatedly applying the following procedure.*

- *If there is a non-root vertex b which does not correspond to a block in $\mathcal{B}$, then identify b with its parent.*

This tree is called the ***relative topology*** *of the blocks in $\mathcal{B}$.*

The notion of relative topology is introduced in order to serve as a succint way of encoding relevant information crossing a small set of vertices. For instance, let X be a v-$W \setminus w$ separator of size at most k for some $v \in V \setminus W$ and $w \in W$. When we focus on the graph induced on $R(v, X) \cup w$, we will need to remember certain information about how the rest of the graph interacts with this subgraph. Since we will be mainly interested in the structure of the *blocks* of $G \setminus S$ (S is a solution) and the relevant information only interacts with this subgraph through $w \cup X$, the notion of relative topologies gives us a way to encode this interaction using only $f(k)$ bits, for some function f. Note that since all but one block of the relative topology are from the given set, the next observation follows.

Observation 7.2. *The number of nodes in the relative topology of ℓ blocks is bounded by $2\ell + 1$.*

We now redefine the notion of well domination but in the context of blocks. This will be crucial for our next step.

Definition 7.3. *Let (G, T, W, k) be a special instance of* Disjoint Subset OCT Compression. *Let $X \subset V \setminus W$, $w \in W$ and v be a vertex disjoint from $W \cup X$ such that,*

- *(X, w) is a minimal v - $W \setminus w$ almost separator in G.*
- *v and w lie in the same block of $G \setminus X$ (which we denote by $\mathbf{B}_X[v, w]$) such that $\mathbf{B}_X[v, w]$ is disjoint from T*

Let $(\hat{X}, w)$ be another almost separator that covers (X, w) and $\mathbf{B}_{\widehat{X}}[v, w]$ is disjoint from T. Then we say that $(\hat{X}, w)$ **well dominates** *(X, w).*

The purpose of the following lemma is to start from a semi-isolated vertex and eventually compute an odd cycle disjoint from a solution or conclude that there is a solution which also happens to be an oct.

LEMMA 7.4. *Consider an instance* $(G = (V, E), T, W, k)$ *of* DISJOINT SUBSET OCT COMPRESSION. *Let* $X \subset V \setminus W$*,* $w \in W$ *and* v *be a vertex disjoint from* $W \cup X$ *such that,*

- (X, w) *is a minimal* v*-*$W \setminus w$ *almost separator*
- v *is semi-isolated from* W *in* $G \setminus X$*.*
- v *and* w *lie in the same block of* $G \setminus X$ *(denoted by* $\mathbf{B}_X[v, w]$*) such that* $\mathbf{B}_X[v, w]$ *is disjoint from* T*.*

Then there is an algorithm running in time $2^{\mathcal{O}(k^2 \log k)} m \log n$ *which returns a set* $\mathcal{R}$ *of* v*-*$W \setminus w$ *almost separators, such that,*

- $s = 2^{\mathcal{O}(k^2 \log k)}$ *and every* Y_i *has size size at most* k*,*
- *Then there is a* $Y \in \mathcal{R}$ *such that* (Y, w) *well dominates* (X, w)*.*

Proof. Let $W_2 = W \setminus w$. Formally, given an instance $I = (G, v, w, W_2, Z, T, k)$, the algorithm returns a set $\mathcal{R}$ of v-W_2 separators of $G \setminus w$ such that,

- $Y_i \subseteq Z$ for every $(Y_i, w) \in \mathcal{R}$.
- Let (X, w) be a v-W_2 almost separator in G, such that $B_X[v, w]$ is T-cycle free in $G \setminus X$. Then $\mathcal{R}$ contains a Y such that (Y, w) is a v-W_2 almost separator in G which well dominates (X, w).

It is clear that the set $\mathcal{R}$ satisfies all the conditions required by the lemma. Here the set Z denotes the set of deletable vertices. We shall narrow down this set as we make recursive calls to the algorithm.

For the sake of a cleaner presentation for the rest of the proof of this lemma, we adopt the convention that whenever we refer to a set being an s-t separator without explicitly mentioning a graph, we mean that the set is an s-t separator in the graph $G \setminus w$ and whenever we refer to a pair being an *almost* s-t separator, we are referring to separation in the graph G. Furthermore, for ease of presentation, we have split up the algorithm and the proof of its correctness and running time into three parts.

Part I: Description of the algorithm.. We first check if there is a v-W_2 separator of size at most k contained in the given subset Z. If not, then we return NO. If there is no path from v to W_2 in $G \setminus w$, then we return $\emptyset$. Otherwise, we compute the tight v-W_2 separator sequence in the graph $G \setminus w$, $\mathcal{I}$, comprising only the vertices in Z. We call a v-W_2 separator P *good* if there is no T-cycle in the block $\mathbf{B}[v, w]$ in the graph $G \setminus P$ and we call it *bad* otherwise. We have the following observation.

OBSERVATION 7.5. *If a* v*-*W_2 *separator is* good*, all* v*-*W_2 *separators covered by this separator are also* good *and if a* v*-*W_2 *separator is* bad*, all* v*-*W_2 *separators which cover this separator are* bad.

Let P_1 be the maximal element of $\mathcal{I}$ which is good and let P_2 be the minimal element of $\mathcal{I}$, which is bad. That is, P_1 is good and every separator in $\mathcal{I} \setminus \{P_1\}$ which covers P_1 is bad, P_2 is bad and every separator in $\mathcal{I} \setminus \{P_2\}$ covered by P_2 is good. If all the separators in $\mathcal{I}$ are good, P_2 is defined as W_2 and if all separators in $\mathcal{I}$ are bad, then P_1 is defined as v. We can compute P_1 and P_2 in time $\mathcal{O}(m + n) \log n$.[2]

[2]We simply need to check for the presence of a terminal in a particular block of a subgraph of G for each separator in $\mathcal{I}$ and we can perform these tests in the form of a binary search over the sequence $\mathcal{I}$.

We now move on to the description of the rest of the algorithm. We have four cases depending on the interaction between X, P_1 and P_2.

1. P_1 covers X in $G \setminus w$.. We add the separator P_1 into the set $\mathcal{R}$.

2. X intersects P_1 or P_2.. For every non-empty $\tilde{S}_1 \subseteq P_1$, we add $\tilde{S}_1$ to each of the separators returned by the recursion on the instance $(G \setminus \tilde{S}_1, v, w, W_2, Z \setminus \tilde{S}_1, T \setminus \tilde{S}_1, k - |\tilde{S}_1|)$ and add the resulting set of separators to $\mathcal{R}$.

Similarly, for every non-empty $\tilde{S}_2 \subseteq P_2$, we add $\tilde{S}_2$ to each of the separators returned by the recursion on $(G \setminus \tilde{S}_2, v, w, W_2, Z \setminus \tilde{S}_2, T \setminus \tilde{S}_2, k - |\tilde{S}_2|)$ and add the resulting set of separators to $\mathcal{R}$.

3. X covers P_1 and P_2 covers X.. We set $Z' = Z \cap (R_{G\setminus w}(v, P_2) \cap NR_{G\setminus w}(v, P_1))$ and add the separators returned by the recursion on (G, v, w, W_2, Z', T, k) to $\mathcal{R}$.

4. X is incomparable with P_1.. Let $G' = G[R(v, P_1) \cup P_1 \cup w]$. Let $P_1^r = P_1 \cap R(v, X \cup w)$ and $P_1^{nr} = P_1 \setminus P_1^r$. We guess how the vertices of P_1^r are divided among the blocks of $G \setminus X$ [3]. Let the blocks which contain a vertex of P_1^r be $\mathbf{B}_1, \ldots, \mathbf{B}_r$. We then make cliques on vertices guessed to be in the same block. We also guess the *relative topology* of these blocks within the block tree of $G[R_{G\setminus w}(v, X) \cup w]$ rooted at block $\mathbf{B}[v, w]$. Let M' be the relative topology and let M be the block tree of $G[R_{G\setminus w}(v, X) \cup w]$ rooted at block $\mathbf{B}[v, w]$.

We add all edges between v, w and the vertices of P_1^r guessed to be in block $\mathbf{B}[v, w]$ in $G \setminus X$. In other words, we make them a clique. Furthermore, for every block $\mathbf{B}_i$, we guess whether or not the cut vertex in H which is the parent of $\mathbf{B}_i$ occurs in the set $U = R(v, P_1 \cup w) \cup w$. If the parent cut vertex of $\mathbf{B}_i$ in M does not occur in U, then we add a vertex corresponding to the parent cut vertex of $\mathbf{B}_i$ in M' (unless we have already added it while guessing for another $\mathbf{B}_i$) and make it adjacent to all vertices in $\mathbf{B}_i$. Finally, if a block $\mathbf{B}_i$ has a child cut vertex in M' which corresponds to a newly added vertex, then we make all the vertices of this block adjacent to this cut vertex. The newly added edges are referred to as *torso edges*. For convenience, we use G' to refer to the constructed graph.[4]

Finally, we construct a new terminal set T' by taking the union of the current terminal set T with the vertices of P_1^r not guessed to be in the block $\mathbf{B}[v, w]$ and all the newly added vertices except the vertices which are the children cut vertices of $\mathbf{B}[v, w]$ to create a new terminal set T'.[5]

Formally, for every possible graph H on the vertices of P_1^r, for every possible rooted block tree M' of H, for every choice (possibly none) of a root block in this block decomposition to be from $\mathbf{B}[v, w]$, we obtain a graph $G_{H,M'}$ and a terminal set T' described as above by considering the chosen rooted block decomposition of H as the relative topology of the corresponding blocks.

Now, for each $G_{H,M'}$, for each $1 \leq k' \leq k - 1$, we recurse on the instance

[3]It is not necessarily a partition as the same vertex can occur in many different blocks.

[4]These constructions are done simply so that we may remember in future recursions, the flow between vertices of P_1^r which is disjoint from the set U.

[5]The intuition behind this modification of the terminal set is that if our guesses were correct, then we are correct to force the newly added vertices (except the children cut vertices of $\mathbf{B}[v, w]$) and the vertices of P_1^r which are in blocks other than $\mathbf{B}[v, w]$ in $G[R(v, X \cup w) \cup w]$ be disjoint from the block containing $\mathbf{B}[v, w]$ at any subsequent point in the recursion.

$(G_{H,M'}, v, w, P_1^{nr}, Z \cap G_{H,M'}, T', k')$. For each separator Y^r returned by this invocation, and for each separator Y^{nr} returned by the invocation $(G \setminus Y^r, v, w, W_2, Z, T, k - |Y^r|)$, add the separator $Y^r \cup Y^{nr}$, to $\mathcal{R}$.

5. X is incomparable with P_2.. We do the same for P_2 as we did for P_1.

Finally, we return the set $\mathcal{R}$.
This completes the description of the algorithm. We now prove the correctness of the algorithm.

Part II: Correctness of the algorithm.. For each tuple $I = (G, v, w, W_2, Z, T, k)$ on which the algorithm is invoked, we define a measure $\mu(I) = 2k - \lambda$ where λ is the size of the smallest v-W_2 separator in the graph $G \setminus w$ which is contained in Z. We prove the correctness of the algorithm by induction on the measure $\mu(I)$.

In the base case, if $\lambda > k$, then the algorithm returns NO, which is clearly correct. Similarly, if $\lambda = 0$, then there is no path between v and W_2 in the graph $G \setminus w$ and hence the algorithm simply returns $\emptyset$ as the separator, which is also correct. This completes the correctness of the base cases and we now assume that the algorithm is correct on all instances I with $\mu(I) < \mu$. Now, consider an instance I such that $\mu(I) = \mu$.

1. P_1 covers the separator X in $G \setminus w$.. Then, the algorithm is correct on the instance I since (P_1, w) itself is a v-W_2 almost separator well dominating (X, w) and the set returned by the algorithm on I contains P_1.

2. X intersects P_1 or P_2.. First we consider the case where X intersects P_1. Let $\tilde{S}_1 = P_1 \cap X$. Let $I' = (G \setminus \tilde{S}_1, v, w, W_2, Z \setminus \tilde{S}_1, T \setminus \tilde{S}_1, k - |\tilde{S}_1|)$. Since P_1 is a minimum v-W_2 separator, we have that the size of the minimum v-W_2 separator in $G \setminus \tilde{S}_1$ is $|P_1| - |\tilde{S}_1|$. Therefore, we have that $\mu(I') = \mu(I) - |\tilde{S}_1| < \mu(I)$. Therefore by the induction hypothesis, the algorithm is correct on I' and hence adding $\tilde{S}_1$ to each separator returned by the invocation on I' returns a set with with required properties with respect to the instance I. The argument for the case when P_1 is disjoint from X but P_2 intersects X is analogous.

3. X covers P_1 and X is covered by P_2.. Let I' be the instance on which the algorithm is recursively invoked in this step. By Lemma 5.11, any v-W_2 separator which lies in the set $NR_{G\setminus w}(v, P_1) \cap R_{G\setminus w}(v, P_2)$ has size at least $|P_1| + 1 = \lambda + 1$. Hence, $\mu(I') \leq \mu - 1$ and by the induction hypothesis, the algorithm on I' returns a set containing a v-W_2 almost separator (X', w) which well dominates (X, w), proving the correctness of the algorithm on the instance I.

4. The separator X is incomparable with P_1.. Let P_1^r be the intersection of P_1 and $R_G(v, (X \cup w))$ and $P_1^{nr} = P_1 \setminus P_1^r$. Also, let X^r be the intersection of X with $R_{G\setminus w}(v, P_1)$ and let X^{nr} be the rest of X. Since X is incomparable with P_1, by Observation 2.7, P_1^r, X^r, P_1^{nr} and X^{nr} are non empty.

Let I' be the instance $(G_{H,M'}, v, w, Z \cap G_H, T', |X^r|)$ where $G_{H,M'}$ is as described in the construction. For convenience we denote $G_{H,M'}$ by G_H. We now prove the following claim which will be used along with the induction hypothesis to prove the correctness of the algorithm on I.

CLAIM 7.6. **(a)** *The block $\mathbf{B}[v, w]$ in $G_H \setminus X^r$ does not contain T'-cycles.*

(b) *Let Y^r be a v-P_1^{nr} separator in G_H such that (Y^r, w) well dominates (X^r, w) in G_H. Then, the set $X' = (X \setminus X^r) \cup Y^r$ is a v-W_2 separator well dominating X in G.*

Proof. For the first statement, suppose that $\mathbf{B}[v, w]$ in $G_H \setminus X^r$ contains a T'-cycle. Consider the T' cycle C which contains v, w and a vertex in T'. If C does not intersect P_1^r, or if it only intersects P_1^r in vertices from the block $\mathbf{B}[v, w]$ in $G \setminus X$, then we can replace the torso edges in C by the corresponding paths in $G \setminus X$ to obtain a closed T-walk C' in $G \setminus X$. But this walk lies in $\mathbf{B}[v, w]$ in $G \setminus X$, which is a contradiction. Hence, C must intersect a vertex of P_1^r which lies outside the block $\mathbf{B}[v, w]$ in $G \setminus X$, that is, C intersects T' in a vertex $t' \in T' \setminus T$. Let $t_1, \ldots, t_r$ be the consecutive vertices of $T' \setminus T$ which appear in C while traversing from t' to v to w and back to t'. We can replace the torso edges of C between each consecutive pair (if the sub-path of C between the pair comprises torso edges) with the corresponding paths which we know are present in $G \setminus X$, to obtain a T'-walk in $G \setminus X$ which contains v and w as well. However, by assumption, the vertices in T' are not in the block $\mathbf{B}[v, w]$ in $G \setminus X$, which is a contradiction. This completes the proof of the first statement.

For the second statement, it is sufficient to show that $\mathbf{B}[v, w]$ in $G \setminus X'$ is T-cycle free. Suppose that $\mathbf{B}[v, w]$ in $G \setminus X'$ contains a T-cycle C, that is, there is a terminal $t \in T$ in the block $\mathbf{B}[v, w]$ in $G \setminus X'$. This implies that there are 2 internally vertex disjoint paths Q_1 and Q_2 in $G \setminus X'$, where Q_1 is a t-v path and Q_2 is a t-w path. Since there is an edge between v and w, we have a T-cycle C containing t, v and w. This cycle cannot intersect P_1^{nr} since X' by definition is a v-P_1^{nr} separator in $G \setminus w$. If C does not contain as sub-paths P_1^r-paths which lie in $NR_{G \setminus w}(v, P_1)$, then C is also present in $\mathbf{B}[v, w]$ in $G_H \setminus Y^r$, a contradiction. Hence, we conclude that C contains P_1^r-paths which lie in $NR_{G \setminus w}(v, P_1)$. However, we can replace these paths by the corresponding torso edges in G_H to get a cycle C' in $G_H \setminus Y^r$ which contains v,w and at least one vertex in T', which is a contradiction. ∎

Now, let Y^r be a v-P_1^{nr} separator in G_J such that (Y^r, w) well dominates (X^r, w) in G_J. Then, due to Claim 7.6, $Y^r \cup X^{nr}$ is a v-W_2 separator which well dominates X in the graph G. Therefore, a set intersecting a v-P_1^{nr} separator which well dominates X^r in the graph G_J also intersects a v-W_2 separator which well dominates X in the graph G. Therefore, if the algorithm is correct on I' and I'', then it is also correct on I and hence it only remains to prove that the algorithm is correct on I' and I''. In order to prove that the algorithm is correct on I' and I'', it is sufficient to prove that $\mu(I'), \mu(I'') < \mu(I) = \mu$ since the correctness then follows from the induction hypothesis.

By Menger's theorem, since P_1 is a minimum size v-W_2 separator, we know that there are P_1^{nr} vertex disjoint paths from v to P_1^{nr}, which is also a lower bound on the size of the smallest v-P_1^{nr} separator in G_J. Now, $\mu(I) = 2(|X^r| + |X^{nr}|) - (|P_1^{nr}| + |P_1^r|)$ and $\mu(I') = 2|X^r| - |P_1^{nr}|$, which implies that $\mu(I') = \mu(I) - (2|X^{nr}| - |P_1^r|)$. Since $|X^{nr}| \geq |P_1^r|$, we have that $\mu(I) - \mu(I') \geq |X^{nr}|$. Since X^{nr} is non empty, we conclude that $\mu(I') < \mu(I)$, which completes the proof of correctness of this case.

5. The separator X is incomparable with P_2. This correctness of this case is analogous to the correctness of the previous case.

We note that the separator X is a good separator by definition and therefore cannot cover P_2 due to Observation 7.5. Therefore the case that X covers P_2 need not be taken into consideration and the cases we have considered are exhaustive. This completes the proof of correctness of the algorithm.

Part III: Running time analysis. We will show by induction on the measure $\mu(I)$ that the number of separators returned by an execution of the algorithm on instance I, $F(\mu(I))$, is bounded by $2^{10\mu(I)^2 \log k}$. In each of the base cases of the algorithm, we either return a single separator of the required kind or say NO. The number of vertices returned in the base case is at most 1, and the claim clearly holds. We now assume that the claimed bound is true for all instances with $\mu(I) < \mu$. Now, consider an instance I such that $\mu(I) = \mu$.

1. The number of separators returned is just 1.

2. There are at most 4^k choices for the sets $\tilde{S}_1$ and $\tilde{S}_2$ and applying the induction hypothesis and summing the number of separators returned in either sub case, the number of separators returned due to Case 1 is bounded by $4^k \cdot 2^{10(\mu-1)^2 \log k}$.

3. By the induction hypothesis, we conclude that the number of separators returned is at most $2^{10(\mu-1)^2 \log k}$.

4. There are at most at most $5^k k^{4k}$ possible graphs we construct and at most k choices for $|X^{nr}|$, and for each of these choices, we return at most $F(\mu_1) \cdot F(\mu_2)$ separators, where $\mu_1 + \mu_2 = \mu$, and hence, applying the induction hypothesis gives us that the number of separators returned is at most $2^{10(\mu-1)^2 \log k} \cdot 2^{10 \log k}$. Hence, the number of separators returned due to case 4 is bounded by $k \cdot 5^k \cdot k^{4k} \cdot 2^k \cdot 2^{10(\mu-1)^2 \log k} \cdot 2^{10 \log k}$.

5. Similarly, the number of separators returned due to case 5 is bounded by $k \cdot 5^k \cdot k^{4k} \cdot 2^{10(\mu-1)^2 \log k} \cdot 2^{10 \log k}$.

Using the fact that $k \leq \mu < 2k$ and $k \geq 2$, we note that the number of separators returned by each case is at most $\frac{1}{8} \cdot 2^{10\mu^2 \log k}$ which yields the required bound. The number of separators returned is hence $2^{\mathcal{O}(k^2 \log k)}$. The algorithm spends $2^{\mathcal{O}(k \log k)}(m+n) + \mathcal{O}((m+n)\log n)$ time at each node of the search tree and hence, the total time taken by the algorithm is $2^{\mathcal{O}(k^2 \log k)} m \log n$. This completes the proof of the lemma. □

Definition 7.7. *The set of almost v-$W \setminus w$ separators returned by the algorithm of Lemma 7.4 are called generalized important v-$W \setminus w$ almost separators. Let (X, w) be a generalized important v-$W \setminus w$ almost separator. The block $\mathbf{B}[v, w]$ in the graph $G \setminus X$ is called an important block. We also denote by $g(k)$ the upper bound on the size of the set of generalized important v-$W \setminus w$ almost separators given by Lemma 7.4.*

We also have the following lemma bounding the number of important blocks containing vertices of a set of bounded size, which also implies the number of important blocks intersecting a particular solution for a given instance of Disjoint Subset OCT Compression.

Lemma 7.8. *The number of important blocks intersecting a set S of size at most k is $2^{\mathcal{O}(k^2 \log k)}$.*

Proof. Consider a vertex v and a generalized important v-$W \setminus w$ almost separator (X, w). We claim that for any vertex u in the block $\mathbf{B}[v, w]$ in $G \setminus X$, (X, w) is also a generalized u-W_2 important almost separator. If this were not the case, then there is a u-W_2 generalized separator (X', w) well dominating (X, w). We claim that (X', w) is also a generalized v-W_2 separator well dominating (X, w). If this were not the case, then there is a T-cycle in the block $\mathbf{B}[v, w]$ in the graph $G \setminus X'$. However, v and u are in the same block as w in $G \setminus X$, implying that they are in the same block as w in $G \setminus X'$. By the definition of (X', w), there is no T-cycle in this block, a contradiction.

Since every vertex occurs in at most $g(k)$ important blocks for a fixed w, the set S intersects at most $k(k+1)g(k)$ important blocks in total, where $g(k) = 2^{\mathcal{O}(k^2 \log k)}$ (by Lemma 7.4). This completes the proof of the lemma. ◻

7.2. Structure of important components and blocks. We now make a structural observation regarding the solution which will be used to simplify the instance further.

Definition 7.9. *Consider an instance $(G = (V, E), T, W, k)$ of* Disjoint Subset OCT Compression. *We call $J \subseteq V$ an* important component *if $G[J]$ is connected and $N(J)$ is an important J-W separator of size at most $k + 1$.*

Lemma 7.10. *Let (G, T, W, k) be a* Yes *instance of* Disjoint Subset OCT Compression. *Then, there is a solution S, such that*

- *for any isolated vertex v with a minimal almost v-W separator $(X, p(X))$ where $X \subseteq S$, $X \cup p(X)$ is an important v-W separator.*
- *for any semi-isolated vertex v with a minimal almost v-$W \setminus w$ separator (X, w) where $X \in S$ and $w \in W$, if $\mathbf{B}[v, w]$ in $G \setminus S$ does not contain T-cycles, then (X, w) is a generalized important v-$W \setminus w$ almost separator.*

Proof. Let S be a T-oct of minimum size which minimizes the number of non-isolated vertices or equivalently, maximizes the number of vertices in $G \setminus S$ which are isolated or semi-isolated. We claim that S satisfies the statement of the lemma. Assume to the contrary that this is not the case.

We now prove the first statement of the lemma. Consider an isolated vertex v such that the set $A = X \cup p(X)$ is not an important v-W separator. Let B be an important v-W separator dominating $X \cup P(X)$. If $p(X) = \emptyset$, then set $p(Y) = \emptyset$ else set $p(Y)$ as any arbitrary vertex of B. Finally, set $Y = B \setminus p(Y)$. We claim that $S' = (S \setminus X) \cup Y$ is a solution with fewer non-isolated vertices, contradicting the minimality of S. Clearly, $|S'| \leq |S|$. Therefore, if S' were not a T-oct, then there is an odd T-cycle in $G \setminus S'$ which intersects a vertex in $X \setminus Y$. But $X \setminus Y$ is separated from W by $Y \cup p(Y)$. Hence, any vertex in $X \setminus Y$ can have at most one vertex disjoint path to W in $G \setminus S'$, a contradiction. It remains to show that the number of non-isolated vertices of S' is strictly less than that of S. Suppose that this is not the case. We first observe that every vertex in $R_G(v, Y \cup p(Y))$ is isolated or semi-isolated in $G \setminus S'$ (follows from the argument which proved that S' is a solution). Now, we show that for any $a \notin Y$, if a is isolated or semi-isolated in $G \setminus S$, it is isolated or semi-isolated in $G \setminus S'$.

Consider a vertex $a \notin Y$ such that a was isolated or semi-isolated in $G \setminus S$. Suppose that a is non-isolated in $G \setminus S'$ and hence has vertex disjoint paths to w_1 and w_2 in $G \setminus S'$. Clearly, one of the two paths must intersect a vertex $u \in X \setminus Y$. This implies that u itself is neither isolated nor semi-isolated. But, $u \in R(v, Y \cup p(Y))$, which is a contradiction since there can be at most 1 vertex disjoint path from u to W. Thus we conclude that the number of non-isolated vertices of S' is at most that of S and since S minimizes the number of non-isolated vertices, the number of non-isolated vertices of S' is equal to that of S.

Observe that if Y contained a non-isolated vertex of S, it implies that S' has strictly less non-isolated vertices. Hence, we assume that Y is disjoint from the set of non-isolated vertices of S. Consider a vertex $z \in Y$.

We first consider the case when $f_{G\setminus S}(z, W) = 0$. In this case, consider the set $S'' = S' \setminus z$. We claim that S'' is also a solution for the given instance. If this were not the case, then there is an odd T-cycle intersecting z. We know that this cycle also intersects a vertex $w \in W$ and a vertex $x \in X \setminus Y$. This implies that x has two vertex disjoint paths to w in $G \setminus S''$. Since $Y \cup p(Y)$ is an x-W separator and the only vertices of $Y \cup p(Y)$ not present in S'' are z and $p(Y)$, it must be the case that the each of the two vertex disjoint x-w paths intersect z and $p(Y)$ respectively. This implies the presence of a z-w path disjoint from $G \setminus S''$. Since this path also avoids X, it is also contained in $G \setminus S$, a contradiction to the assumption that $f_{G\setminus S}(z, W) = 0$. Hence, we conclude that S'' is indeed a solution. However, S'' is a solution which is smaller than S, which is a contradiction to the optimality of S.

We now consider the case where $f_{G\setminus S}(z, W) \geq 1$. Then, there is a vertex u in the block B_W in $G \setminus S$ such that $f_{G\setminus(S\cup u)}(z, W \setminus u) = 0$, that is every path from z to W intersects u and u is a non-isolated vertex with respect to S. Consider the set $S'' = (S' \setminus z) \cup u$. We claim that S'' is also a solution for the given instance. If this were not the case, then there is an odd T-cycle intersecting z. We know that this cycle also intersects a vertex $w \in W$ and a vertex $x \in X \setminus Y$. This implies that x has two vertex disjoint paths to w in $G \setminus S''$. Since $Y \cup p(Y)$ is an x-W separator and the only vertices of $Y \cup p(Y)$ not present in S'' are z and $p(Y)$, it must be the case that the each of the two vertex disjoint x-w paths intersect z and $p(Y)$ respectively. This implies the presence of a z-w path disjoint from $G \setminus S''$. Since this path also avoids X, it is also contained in $G \setminus S$, in which case it intersects u, which is contained in S''. Hence, we conclude that S'' is indeed a solution. However, since S'' is an optimum solution which intersects the non-isolated set of S, S'' has fewer non-isolated vertices, a contradiction. This completes the proof of the first statement of the lemma.

We now move on to the second statement. Consider a semi-isolated vertex v such that the set $X \cup w$ is not a generalized important v-W_2 separator. Let (Y, w) be a generalized important v-W_2 separator well dominating (X, w). Let z be any vertex of $Y \setminus X$. We claim that the set $S' = (S \setminus X) \cup (Y \setminus z) \cup w$ is also a solution.

If S' were not a solution, then there is an odd T-cycle intersecting a vertex in $X \setminus Y$. But, $X \setminus Y$ is separated from W_2 by $Y \cup w$. Hence, any vertex in $X \setminus Y$ can have at most 1 vertex disjoint path to W_2 in $(Y \setminus z) \cup w$, which implies the presence of at most 1 vertex disjoint $X \setminus Y$-W_2 path in $G \setminus S'$. This contradicts our assumption that there was an odd T-cycle in $G \setminus S'$ intersecting W_2 and $X \setminus Y$. Hence, we conclude that S' is also a solution. However, S' is an optimum solution which intersects W a contradiction to the definition of the DISJOINT SUBSET OCT COMPRESSION problem, which promises that there are no optimum solutions intersecting W. This completes the proof of the lemma. ∎

Unless explicitly mentioned otherwise, we assume any solution we consider henceforth to any given instance of DISJOINT SUBSET OCT COMPRESSION to be of the kind described in the proof of Lemma 7.10.

7.3. Reducing the subset variant to the standard variant.

LEMMA 7.11. *There is an algorithm that, given an instance (G, T, W, k) of* DISJOINT SUBSET OCT COMPRESSION, *runs in time $2^{\mathcal{O}(k^2 \log k)} mn \log n$ and returns a set of $2^{\mathcal{O}(k^2 \log k)}$ vertices which contain an isolated or semi-isolated with respect to some solution or concludes correctly either that the given instance has a solution* *which is also an odd cycle transversal for G or that the given instance has no solution.*

Proof. Let $\mathcal{Z}$ be the union of the set $\mathcal{Z}_1$ of all important components J in G and the set $\mathcal{Z}_2$ of all generalized important blocks which correspond to a generalized important separator of size at most $k+1$. Let $\mathcal{Z}_o$ be the set of all components J in $\mathcal{Z}_1$ such that $G[J \cup v]$ contains an odd cycle for some $v \in N(J)$ and the set of all blocks J in $\mathcal{Z}_2$ such that $G[J]$ contains an odd cycle. By Lemma 2.10 and Lemma 7.4, $\mathcal{Z}$ and $\mathcal{Z}_o$ can be computed in time $2^{\mathcal{O}(k^2 \log k)} mn \log n$. If $|\mathcal{Z}_o| > k4^{k+1} + k(k+1)g(k)$, then we set $\mathcal{Z}'_o$ as any subset of $\mathcal{Z}_o$ of size $k4^{k+1} + k(k+1)g(k) + 1$ and set $\mathcal{Z}'_o = \mathcal{Z}_o$ otherwise.

We now construct a set $\mathcal{R}$ as follows. For each component or block $J \in \mathcal{Z}'_o$, add $N(J)$ to $\mathcal{R}$. Since the neighborhood of each component or block in $\mathcal{Z}'_o$ is bounded by $k+1$, we will add $2^{\mathcal{O}(k^2 \log k)}$ vertices to $\mathcal{R}$. Now, for each $J \in \mathcal{Z}'_o$, we pick an arbitrary cycle in $G[J \cup N(J)]$ and pick two arbitrary vertices of this cycle which lies in J and add it to $\mathcal{R}$. Since we add a single vertex for each element of $\mathcal{Z}'_o$, the size of $\mathcal{R}$ is still $2^{\mathcal{O}(k^2 \log k)}$. Finally we return $\mathcal{R}$. Observe that the size of the set $\mathcal{R}$ and the time taken to construct it follow from Lemma 2.10 and Lemma 7.4. It only remains for us to to show that $\mathcal{R}$ has the requisite properties. Suppose that the given instance is a Yes instance of Type 2, $\mathcal{R}$ is disjoint from any solution and there is no solution for the given instance which is also an odd cycle transversal in G.

Consider any solution S of Type 2 and consider the graph $G \setminus S$. Since S is not an odd cycle transversal, there is an odd cycle in $G \setminus S$. Furthermore, since S is a solution of Type 2, we have by Lemma 5.2 that the block $\mathbf{B}[W]$ in $G \setminus S$ is bipartite. Therefore, we conclude that any odd cycle in $G \setminus S$ has at most one vertex in the block $\mathbf{B}[W]$.

Case 1: $|\mathcal{Z}_o| > k4^{k+1} + k(k+1)g(k)$. In this case, by Lemma 2.10 and Lemma 7.8, the number of components or blocks of $\mathcal{Z}_o$ intersecting S is at most $k4^{k+1} + k(k+1)g(k)$, implying that at least one of the components or blocks in $\mathcal{Z}'_o$, say J, must be disjoint from S. Now, consider any odd cycle C in $G[J \cup v]$ or $G[J]$ depending on whether J is an important component or a block, where $v \in N(J)$. Since $\mathcal{R}$ is disjoint from any solution, $N(J)$ is disjoint from any solution, implying that C is an odd cycle in $G \setminus S$. Since at most one vertex of C can be in $\mathbf{B}[W]$, of any arbitrary pair of vertices of C, at least one vertex is isolated or semi-isolated with respect to S and hence $\mathcal{R}$ contains an isolated or semi-isolated vertex with respect to some solution.

Case 2: $|Z_o| \leq k4^{k+1} + k(k+1)g(k)$. In this case, if there is an odd cycle disjoint from S, it occurs in an important component or block and hence it occurs in $\mathcal{Z}_o$. However, we have that $\mathcal{Z}'_o = \mathcal{Z}_o$. Therefore, this component or block also occurs in $\mathcal{Z}_o$. The rest of the argument for this case is identical to that of the previous case, thus proving that $\mathcal{R}$ indeed has the properties claimed in the statement of the lemma. This completes the proof of Lemma 3.3. ∎

7.4. Algorithm for special instances of Type 2.

We are now ready to present the full algorithm to detect special solutions of Type 2. We begin by recalling the following algorithm for solving odd cycle transversal.

Lemma 7.12. *Let (G, T, W, k) be an instance* Disjoint Subset OCT Compression. *Then, there is an algorithm running in time $(4^{k+\mathcal{O}(\log k)}(m+n))$ which returns a solution or concludes correctly that there is no solution which is also an odd cycle transversal of G.*

Proof. The lemma follows from the observation that (G, T, W, k) is a Yes instance of Subset OCT if and only if (G, k) is a Yes instance of OCT and the OCT algorithm of [29, 14]. □

We are now ready to summarize our algorithm for special instances of Type 2.

LEMMA 7.13. *There is an algorithm that, given an instance (G, T, W, k) of* DISJOINT SUBSET OCT COMPRESSION, *runs in time $2^{\mathcal{O}(k^3 \log k)} mn \log n$ and either returns a solution for the given instance or correctly concludes that no Type 2 solution exists.*

Proof. We first make a complete bipartite graph on a bipartition of W which is consistent with that of the block $\mathbf{B}[W]$ in the graph $G \setminus S$ where S is some solution of Type 2. There are $2^{|W|}$ such possible bipartitions and we apply the steps below for each such bipartition after making a complete bipartite graph on W based on the chosen bipartition.

We then apply the algorithm given by Lemma 3.3 on the given instance I. If this algorithm returns a non-empty set $\mathcal{R}$, then we know that it either intersects a solution or contains an isolated or a semi-isolated vertex, in which case, for each vertex, we either directly branch on it, or run the algorithms of Lemma 5.1 and Lemma 5.17 and branch on the vertices returned by the algorithm of each lemma. Since we have made a complete biparition on W, the resulting instances will remain special instances of Type 2, allowing further recursion.

Otherwise, if the algorithm of Lemma 3.3 concludes that either there is no solution or there is a solution for the given instance which is also an odd cycle transversal, then we apply Lemma 7.12 to find a solution for the instance.

The correctness as well as claimed the running time of this algorithm is implied by those of Lemmas 5.1, 5.17, 3.3, and 7.12. This concludes the algorithm to solve special instances of DISJOINT SUBSET OCT COMPRESSION of Type 2. □

8. Solving DISJOINT SUBSET OCT COMPRESSION on general instances.

We now consider the case that for the given instance (G, T, W, k) of DISJOINT SUBSET OCT COMPRESSION, any solution S is such that there is no block in $G \setminus S$ which contains all the vertices in W. In this case, let $W = W_1 \uplus W_2$ be a partition such that the vertices in W_1 occur in a block, say B in $G \setminus S$ and there is at most one vertex, say p in $V \setminus S$ which separates W_1 from W_2 in $G \setminus S$. Observe that this is the same as saying that W_1 is semi-isolated from $W_2 \cup p$ if $p \notin W_1$ and $W_1 \setminus p$ is semi-isolated from $W_2 \cup p$ otherwise.

8.1. Algorithm for the case when p is known.

OBSERVATION 8.1. *Given a* YES *instance (G, T, W, k) of* DISJOINT SUBSET OCT COMPRESSION, *let S be a solution for this instance. Let $W_1 \uplus W_2$ be a partition of W such that W_1 occurs in a block and there is a set $X \subseteq S$ which, along with at most one other vertex which is disjoint from $S \cup W$, say p, separates W_1 from W_2 in $G \setminus S$. Let $U = R(W_1, X \cup p) \cup p$, $G' = G[U]$ and $T' = T \cap U$. Then, $S' = S \cap U$ is a special solution for the instance $(G', T', W_1, |S \cap U|)$.*

In this case, we can apply Lemma 5.12 by setting $Q_1 = W_1 \setminus p$ and $Q_2 = W_2$ and $\mathsf{piv} = p$. Recall that we also need the algorithm $\mathcal{A}$ to find a minimum T-oct in the graph $G' = G[R(Q_1, \hat{X} \cup \mathsf{piv}) \cup \mathsf{piv}]$ for any Q_1-Q_2 almost separator $(\hat{X}, \mathsf{piv})$. However, by Observation 8.1, there is a special solution for the instance induced on the G', which implies that the algorithm $\mathcal{A}$ only has to look for special solutions, hence allowing us to use the algorithms of Lemma 3.2 or Lemma 3.4 as the algorithm $\mathcal{A}$. Therefore, we infer the following lemma.

LEMMA 8.2. *Given a* YES *instance* (G, T, W, k) *of* DISJOINT SUBSET OCT COMPRESSION*, let S be a solution for this instance. Let $W_1 \uplus W_2$ be a partition of W such that W_1 occurs in a block and there is a set $X \subseteq S$ which, along with at most one other vertex which is disjoint from $S \cup W$, say p, separates W_1 from W_2 in $G \setminus S$. There is an algorithm that, given W_1, p and W_2, runs in time $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$ and returns a set of $2^{\mathcal{O}(k^2)}$ vertices which*

- *intersects some solution for the given instance, or*
- *contains a vertex v which is isolated or semi-isolated with respect to some solution for the given instance.*

By combining this lemma with the algorithms of Lemma 5.1 and Lemma 5.17, we get Lemma 3.5.

LEMMA 8.3. *Given a* YES *instance* (G, T, W, k) *of* DISJOINT SUBSET OCT COMPRESSION*, let S be a solution for this instance. Let $W_1 \uplus W_2$ be a partition of W such that W_1 occurs in a block and there is a set $X \subseteq S$ which, along with at most one other vertex which is disjoint from $S \cup W$, say p, separates W_1 from W_2. There is an algorithm that, given W_1, p and W_2, runs in time $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$ and returns a set of $2^{\mathcal{O}(k^2)}$ vertices which intersects a solution for the given instance.*

8.2. Algorithm for the case when p is unknown. In this case, our strategy is to modify the algorithm of Lemma 5.12 to also account for the fact that the vertex piv in the statement of the lemma is not given to the algorithm. The proof of this lemma is identical to that of Lemma 5.12 except for 2 minor differences, which we mention.

LEMMA 8.4. *Given a* YES *instance* (G, T, W, k) *of* DISJOINT SUBSET OCT COMPRESSION*, let S be a solution for this instance. Let $W_1 \uplus W_2$ be a partition of W such that W_1 occurs in a block in $G \setminus S$ and there is a set $X \subseteq S$ which, along with at most one other vertex which is disjoint from $S \cup W$, say p, separates W_1 from W_2. There is an algorithm that, given W_1 and W_2, runs in time $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$ and returns a set of $2^{\mathcal{O}(k^2)}$ vertices which*

- *intersects some solution for the given instance, or*
- *contains p or*
- *contains a vertex v which is isolated or semi-isolated with respect to some solution for the given instance.*

Proof. (sketch) After we compute a tight sequence $\mathcal{I}$ of W_1-W_2 separators of size at most $k + 1$, we need a notion of *good* and *bad* separators. In this case, we define a good separator as a separator $P \in \mathcal{I}$ which has the property that there is a vertex $x \in P$ such that there is a T-oct of size at most ℓ in the graph $G[R(W_1, P) \cup x]$ and bad separators are separators which do not possess this property. As before we iterate over all choices of $\ell \in [1, k]$.

The second difference is when we construct sub-instances and recurse on them (see Step 3 in the algorithm of Lemma 5.12). We give a brief description of this step for the current algorithm. Suppose that P_1 is the maximal good separator in $\mathcal{I}$. Let $P_1^r = P_1 \cap (R(W_1, X \cup p))$ and $P_1^{nr} = P_1 \setminus P_1^r$. Let $X^r = X \cap R(W_1, P)$. As before we can assume that P_1^r lies in the same block as Q_1 in $G \setminus S$. We now consider the following 2 cases.

$p \notin R(W_1, P)$**:** In this case, we build the graph G' by taking the graph $G[R(W_1, P) \cup P]$ and making a clique or a (appropriately guessed) bipartition on the set $W_1 \cup P_1^r$ since these vertices are guessed to be part of the block $\mathbf{B}[W_1]$ in $G \setminus S$. We then run the algorithm of Lemma 3.5 on the graph G' with the set W_1 being the same as our "current" W_1, W_2 set to P_1^{nr}, p as the empty set and X^r considered as X for G', and return the set of vertices returned by this algorithm.

$p \in R(W_1, P)$: In this case, we build the graph G' as above and recurse on this graph with W_2 now set as P_1^{nr} (with X^r considered as X) while p remains unknown.

We note that in both cases $|X^r| < |X|$ as X^{nr} is non-empty. Thus k decreases in each sub-instance. Thus the measure $\mu = 2k - \lambda$ also drops. The proof of correctness and running time are both the same as that of Lemma 5.12.

Finally, we combine Lemma 3.6 with Lemma 3.5, Lemma 5.1 and Lemma 5.17 to get Lemma 3.7.

LEMMA 8.5. *Given a* YES *instance* (G, T, W, k) *of* DISJOINT SUBSET OCT COMPRESSION*, let S be a solution for this instance. Let $W_1 \uplus W_2$ be a partition of W such that W_1 occurs in a block in $G \setminus S$ and there is a set $X \subseteq S$ which, along with at most one other vertex which is disjoint from $S \cup W$, say p, separates W_1 from W_2. There is an algorithm that, given W_1 and W_2, runs in time $2^{\mathcal{O}(k^3 \log k)} mn \log^2 n$ and returns a set of $2^{\mathcal{O}(k^2)}$ vertices which intersects a solution for the given instance.*

Since the above lemma can be used along with the other subroutines to solve general instances of DISJOINT SUBSET OCT COMPRESSION, this completes the description of our algorithm for DISJOINT SUBSET OCT COMPRESSION and by our discussion in Section 3, the proof of Lemma 3.1.

9. Conclusion.

In this paper, we gave the first FPT algorithm for the SUBSET OCT problem where the exponential dependence of the running time of the algorithm on k is polynomial. Our algorithm avoids the machinery of graphs minors and is based entirely on separation and flow based arguments and improves upon the algorithm of Kakimura et al. with respect to both the parameter as well as the input size.

Interesting problems in this direction include obtaining an algorithm for the above problem where the function of the parameter is $2^{\mathcal{O}(k)}$ as well as an improved algorithm for the subset version of EVEN CYCLE TRANSVERSAL. Finally, we leave open the question of obtaining an FPT algorithm with a linear dependence on the input size.

REFERENCES

[1] N. BOUSQUET, J. DALIGAULT, AND S. THOMASSÉ, *Multicut is fpt*, in STOC, 2011, pp. 459–468.
[2] J. CHEN, Y. LIU, AND S. LU, *An improved parameterized algorithm for the minimum node multiway cut problem*, Algorithmica, 55 (2009), pp. 1–13.
[3] J. CHEN, Y. LIU, S. LU, B. O'SULLIVAN, AND I. RAZGON, *A fixed-parameter algorithm for the directed feedback vertex set problem*, J. ACM, 55 (2008).
[4] R. H. CHITNIS, M. CYGAN, M. T. HAJIAGHAYI, AND D. MARX, *Directed subset feedback vertex set is fixed-parameter tractable*, ACM Transactions on Algorithms, 11 (2015), p. 28.
[5] R. H. CHITNIS, M. HAJIAGHAYI, AND D. MARX, *Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset*, SIAM J. Comput., 42 (2013), pp. 1674–1696.
[6] M. CYGAN, F. V. FOMIN, L. KOWALIK, D. LOKSHTANOV, D. MARX, M. PILIPCZUK, M. PILIPCZUK, AND S. SAURABH, *Parameterized Algorithms*, Springer, 2015, https://doi.org/10.1007/978-3-319-21275-3, http://dx.doi.org/10.1007/978-3-319-21275-3.
[7] M. CYGAN, M. PILIPCZUK, M. PILIPCZUK, AND J. O. WOJTASZCZYK, *On multiway cut parameterized above lower bounds*, TOCT, 5 (2013), p. 3.
[8] M. CYGAN, M. PILIPCZUK, M. PILIPCZUK, AND J. O. WOJTASZCZYK, *Subset feedback vertex set is fixed-parameter tractable*, SIAM J. Discrete Math., 27 (2013), pp. 290–309.
[9] R. DIESTEL, *Graph Theory, 4th Edition*, vol. 173 of Graduate texts in mathematics, Springer, 2012.

[10] S. Fiorini, N. Hardy, B. A. Reed, and A. Vetta, *Planar graph bipartization in linear time*, Discrete Applied Mathematics, 156 (2008), pp. 1175–1180.

[11] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin, 2006.

[12] J. Hopcroft and R. Tarjan, *Algorithm 447: efficient algorithms for graph manipulation*, Commun. ACM, 16 (1973), pp. 372–378, https://doi.org/10.1145/362248.362272, http://doi.acm.org/10.1145/362248.362272.

[13] F. Hüffner, *Algorithm engineering for optimal graph bipartization*, J. Graph Algorithms Appl., 13 (2009), pp. 77–98.

[14] Y. Iwata, K. Oka, and Y. Yoshida, *Linear-time fpt algorithms via network flow*, in SODA, 2014, pp. 1749–1761.

[15] N. Kakimura, K. Kawarabayashi, and Y. Kobayashi, *Erdös-pósa property and its algorithmic applications: parity constraints, subset feedback set, and subset packing*, in SODA, 2012, pp. 1726–1736.

[16] K. Kawarabayashi and Y. Kobayashi, *Fixed-parameter tractability for the subset feedback set problem and the s-cycle packing problem*, J. Comb. Theory, Ser. B, 102 (2012), pp. 1020–1034.

[17] K. Kawarabayashi and B. A. Reed, *An (almost) linear time algorithm for odd cycles transversal*, in SODA, 2010, pp. 365–378.

[18] S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström, *Fixed-parameter tractability of multicut in directed acyclic graphs*, SIAM J. Discrete Math., 29 (2015), pp. 122–144.

[19] D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh, *Faster parameterized algorithms using linear programming*, ACM Transactions on Algorithms, 11 (2014), pp. 15:1–15:31.

[20] D. Lokshtanov and M. S. Ramanujan, *Parameterized tractability of multiway cut with parity constraints*, in ICALP (1), 2012, pp. 750–761.

[21] D. Lokshtanov, M. S. Ramanujan, and S. Saurabh, *Linear time parameterized algorithms for subset feedback vertex set*, in Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I, vol. 9134 of Lecture Notes in Computer Science, Springer, 2015, pp. 935–946.

[22] D. Lokshtanov, S. Saurabh, and S. Sikdar, *Simpler parameterized algorithm for oct*, in IWOCA, 2009, pp. 380–384.

[23] D. Lokshtanov, S. Saurabh, and M. Wahlström, *Subexponential parameterized odd cycle transversal on planar graphs*, in FSTTCS, 2012, pp. 424–434.

[24] D. Marx, *Parameterized graph separation problems*, Theoret. Comput. Sci., 351 (2006), pp. 394–406.

[25] D. Marx, B. O'Sullivan, and I. Razgon, *Finding small separators in linear time via treewidth reduction*, ACM Transactions on Algorithms, 9 (2013), p. 30.

[26] D. Marx and I. Razgon, *Fixed-parameter tractability of multicut parameterized by the size of the cutset*, SIAM J. Comput., 43 (2014), pp. 355–388.

[27] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press, Oxford, 2006.

[28] V. Raman, M. S. Ramanujan, and S. Saurabh, *Paths, flowers and vertex cover*, in ESA, vol. 6942 of Lecture Notes in Computer Science, 2011, pp. 382–393.

[29] M. S. Ramanujan and S. Saurabh, *Linear time parameterized algorithms via skew-symmetric multicuts*, in SODA, 2014, pp. 1739–1748.

[30] I. Razgon and B. O'Sullivan, *Almost 2-sat is fixed-parameter tractable.*, J. Comput. Syst. Sci., 75 (2009), pp. 435–450, http://dblp.uni-trier.de/db/journals/jcss/jcss75.html#RazgonO09.

[31] B. A. Reed, K. Smith, and A. Vetta, *Finding odd cycle transversals*, Oper. Res. Lett., 32 (2004), pp. 299–301.

[32] R. E. Tarjan, *Data structures and network algorithms*, vol. 44 of CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1983, https://doi.org/10.1137/1.9781611970265, http://dx.doi.org/10.1137/1.9781611970265.