SIAM J. DISCRETE MATH.
Vol. 34, No. 2, pp. 1302–1325

PATH CONTRACTION FASTER THAN 2^n*

AKANKSHA AGRAWAL[†], FEDOR V. FOMIN[‡], DANIEL LOKSHTANOV[§], SAKET SAURABH[¶], AND PRAFULLKUMAR TALE[‖]

Abstract. A graph G is contractible to a graph H if there is a set $X \subseteq E(G)$, such that G/X is isomorphic to H. Here, G/X is the graph obtained from G by contracting all the edges in X. For a family of graphs $\mathcal{F}$, the $\mathcal{F}$-CONTRACTION problem takes as input a graph G on n vertices, and the objective is to output the largest integer t, such that G is contractible to a graph $H \in \mathcal{F}$, where $|V(H)| = t$. When $\mathcal{F}$ is the family of paths, then the corresponding $\mathcal{F}$-CONTRACTION problem is called PATH CONTRACTION. The problem PATH CONTRACTION admits a simple algorithm running in time $2^n \cdot n^{\mathcal{O}(1)}$. In spite of the deceptive simplicity of the problem, beating the $2^n \cdot n^{\mathcal{O}(1)}$ bound for PATH CONTRACTION seems quite challenging. In this paper, we design an exact exponential time algorithm for PATH CONTRACTION that runs in time $1.99987^n \cdot n^{\mathcal{O}(1)}$. We also define a problem called 3-DISJOINT CONNECTED SUBGRAPHS and design an algorithm for it that runs in time $1.88^n \cdot n^{\mathcal{O}(1)}$. The above algorithm is used as a subroutine in our algorithm for PATH CONTRACTION.

Key words. path contraction, exact exponential time algorithms, graph algorithms, enumerating connected sets, 3-disjoint connected subgraphs

AMS subject classifications. 68R10, 68R01, 05C85

DOI. 10.1137/19M1259638

1. Introduction.

Graph editing problems are one of the central problems in graph theory that have received a lot of attention in algorithm design. Some of the natural graph editing operations are vertex/edge deletion, edge addition, and edge contraction. For a family of graphs $\mathcal{F}$, the $\mathcal{F}$-EDITING problem takes as input a graph G, and the objective is to find the minimum number of operations required to transform G into a graph from $\mathcal{F}$. In fact, the $\mathcal{F}$-EDITING problem, where the edit operations are restricted to one of vertex deletion, edge deletion, edge addition, or edge contraction have also received a lot of attention in algorithm design. The $\mathcal{F}$-EDITING problems encompass several classical NP-hard problems like VERTEX COVER, FEEDBACK VERTEX SET, LONGEST PATH, etc.

The $\mathcal{F}$-EDITING problem where the only allowed edit operation is edge contraction, is called $\mathcal{F}$-CONTRACTION. For a graph G and an edge $e = uv \in E(G)$, *contraction* of an edge uv in G results in a graph G/e, which is obtained by deleting u and v from G, adding a new vertex w_e and making w_e adjacent to the neighbors of u or v (other than u, v). A graph G is *contractible* to a graph H, if there exists a subset $X \subseteq E(G)$, such that if we contract each edge from X, then the resulting graph is

*Received by the editors May 2, 2019; accepted for publication (in revised form) April 15, 2020; published electronically June 11, 2020. A preliminary version of this work has appeared in *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming* (ICALP 2019).
https://doi.org/10.1137/19M1259638
Funding: The research leading to these results has been supported by the Research Council of Norway via the project MULTIVAL. The first author is supported by the PBC Program of Fellowships for Outstanding Post-doctoral Researchers from China and India (grant 5101479000), and the fourth author is supported by the Horizon 2020 Framework program, ERC Consolidator Grant LOPPRE (grant 819416).
[†]Ben-Gurion University of the Negev, Beersheba 84105, Israel (agrawal@post.bgu.ac.il).
[‡]University of Bergen, P.O.Box 7800, 5020 Bergen, Norway (fomin@ii.uib.no).
[§]University of California Santa Barbara, Santa Barbara, CA 93106 (daniello@ucsb.edu).
[¶]Institute of Mathematical Sciences, HBNI, Chennai 600 113, India; University of Bergen, P.O.Box 7800, 5020 Bergen, Norway; and UMI ReLax, Chennai, India (saket@imsc.res.in).
[‖]Institute of Mathematical Sciences, HBNI, Chennai 600 113, India (pptale@imsc.res.in).

isomorphic to H. For several families of graphs $\mathcal{F}$, early papers by Watanabe, Ae, and Nakamura [18, 19] and Asano and Hirata [1] showed that $\mathcal{F}$-Contraction is NP-hard. The NP-hardness of problems like Tree Contraction and Path Contraction, which are the $\mathcal{F}$-Contraction problems for the family of trees and paths, respectively, follows easily from [1, 3]. A restricted version of Path Contraction is the problem P_t-Contraction, where t is a fixed constant. P_t-Contraction is shown to be NP-hard even for $t = 4$, while for $t \leq 3$, the problem is polynomial time solvable [3]. P_t-Contraction alone had received lot of attention for smaller values of t, even when the input graph is from a very structured family of graphs (for instance, see [3, 17, 10, 6, 8, 13] and the references therein).

Several NP-hard problem like SAT, k-SAT, Vertex Cover, Hamiltonian Path, etc., are known to admit an algorithm running in time $\mathcal{O}^\star(2^n)$.[1] These results are obtained by techniques like brute force search, dynamic programming over subsets, etc. One of the main questions that arise in this context is whether we can break the $\mathcal{O}^\star(2^n)$ barrier for these problems. In fact, the hardness of SAT gives rise to the strong exponential time hypothesis (SETH) of Impagliazzo and Paturi [12, 11], which rules out existence of an $\mathcal{O}^\star((2-\epsilon)^n)$-time algorithm for SAT, for any $\epsilon > 0$. SETH has been used to obtain such algorithmic lower bounds for many other NP-hard problems (see, for example, [4, 14]). Not all NP-hard problems seem to be as "hard" as SAT. For many NP-hard problems, it is possible to break the $\mathcal{O}^\star(2^n)$ barrier. For instance, problems like Vertex Cover and (undirected) Hamiltonian Path are known to admit algorithms running in time $\mathcal{O}^\star((2-\epsilon)^n)$, for some $\epsilon > 0$ [2, 15]. Thus, a natural question is for which NP-hard problems can we avoid the "brute force search" and obtain algorithms that are better than $\mathcal{O}^\star(2^n)$.

In this article, we focus on the problem Path Contraction, which is formally defined below.

Path Contraction
Input: Graph G.
Output: Largest integer t, such that G is contractible to P_t.

Note that if t is the largest integer such that G is contractible to P_t, then the minimum number of edge contraction operations is $(n-t)$. Path Contraction is known to admit a simple algorithm that runs in time $\mathcal{O}^\star(2^n)$. Such an algorithm can be obtained by coloring the input graph with two colors and contracting connected components in the colored subgraphs. For a deceptively simple problem like Path Contraction, it seems quite challenging to break the $\mathcal{O}^\star(2^n)$ barrier. The problem 2-Disjoint Connected Subgraphs (2-DCS)[2] can be "roughly" interpreted as solving P_4-Contraction. (We can use the algorithm for 2-DCS to solve P_4-Contraction within the same time bound.) There have been studies, which break the $\mathcal{O}^\star(2^n)$ brute force barrier, for 2-DCS. In particular, Cygan et al. [5] designed an $\mathcal{O}^\star(1.933^n)$ algorithm for 2-DCS. This result was improved by Telle and Villanger, who designed an algorithm running in time $\mathcal{O}^\star(1.7804^n)$ for the problem [16]. The main goal of this article is to break the $\mathcal{O}^\star(2^n)$ barrier for Path Contraction. Obtaining such an algorithm for Path Contraction was stated as an open problem in [17].

Our results. We design an algorithm for Path Contraction running in time $\mathcal{O}^\star(1.99987^n)$, where n is the number of vertices in the input graph. To the best of

[1]The $\mathcal{O}^\star$ notation hides polynomial factors in the running time expression.

[2]See section 3 for formal definitions.

our knowledge, this is the first nontrivial algorithm for the problem, which breaks the $\mathcal{O}^\star(2^n)$ barrier. To obtain our main algorithm for Path Contraction, we design four different algorithms for the problem, which are used as subroutines to the main algorithm. We exploit the property that certain types of algorithms are better for certain instances but may be inefficient for certain other instances. Roughly speaking, we look for solutions using different algorithms, and then the best suited algorithm for the instance is used to return the solution. When one of the four algorithms is called as a subroutine, it does not necessarily return an optimum solution for the instance; rather it only looks for solutions that satisfy certain conditions. These conditions are quantified by fractions associated with the input graph. We note that for appropriate values of these "fractions," each of our subroutines still serves as an algorithm for Path Contraction (and thus can compute the optimal solution). We argue that there is always a solution which satisfies the conditions for one of the subroutines, by setting the values of the fractions appropriately. A saving over $\mathcal{O}^\star(2^n)$, in the running time achieved by our algorithm, also exploits the property that "small" connected sets with bounded neighborhood can be enumerated "efficiently."

In the following we very briefly explain the type of solutions we look for, in our subroutines. Consider a path P_t, such that G can be contracted to P_t, where t is the largest such integer. The solution t can be "witnessed" by a partition $\mathcal{W} = \{W_1, W_2, \ldots, W_t\}$ of $V(G)$, where the vertices from W_i "merge" to the ith vertex of P_t (a formal definition for it can be found in section 2). Such a "witness" is called a P_t-*witness structure*. The first (subroutine) algorithm for Path Contraction searches for a solution where the P_t-witness structure can be "split" into two connected disjoint parts which are "small." Then, it exploits the "smallness" of the parts to compute solutions efficiently and combines them to compute the solution for the whole graph. The second subroutine searches for a pair of sets in the P_t-witness structure which are very dense. Then it exploits the sparseness of the remaining graphs to efficiently compute partial solutions for them. Moreover, the pair of dense parts are resolved using the algorithm of Telle and Villanger for 2-Disjoint Connected Subgraph [16]. The third routine works with a hope that the total number of vertices in one of the odd/even sets from $\mathcal{W}$ can be bounded. Finally, the fourth subroutine works by exploiting a similar odd/even property as the third subroutine, but it relaxes the condition to "nearly" small odd/even set.

To design our algorithm, we also define a problem called 3-Disjoint Connected Subgraphs (3-DCS), which is an extension of the 2-Disjoint Connected Subgraphs (2-DCS) problem. 3-DCS takes as input a graph G and disjoint sets $Z_1, Z_2 \subseteq V(G)$, and the goal is to partition $V(G)$ into three sets (V_1, U, V_2), such that graphs induced on each of the parts are connected and $Z_i \subseteq V_i$ for $i \in [2]$. We design an algorithm for 3-DCS running in time $\mathcal{O}^\star(1.88^n)$. The fourth subroutine of our algorithm uses the algorithm for 3-DCS as a subroutine.

As a corollary to our $\mathcal{O}^\star(1.88^n)$-time algorithm for 3-DCS, we obtain that P_5-Contraction admits an algorithm running in time $\mathcal{O}^\star(1.88^n)$.

2. Preliminaries.

In this section, we state some basic definitions and introduce terminologies from graph theory. We use standard terminology from the book of Diestel [7] for the graph related terminologies which are not explicitly defined here. We also establish some notation that will be used throughout.

We denote the set of natural numbers by $\mathbb{N}$ (including 0). For $k \in \mathbb{N}$, $[k]$ denotes the set $\{1, 2, \ldots, k\}$.

We note that all graphs considered in this article are connected graphs on at least two vertices (unless stated otherwise). For a graph G, the sets $V(G)$ and $E(G)$ denote the sets of vertices and edges in G, respectively. Two (distinct) vertices u, v in $V(G)$ are *adjacent* if the edge $uv \in E(G)$. For an edge uv, the vertices u and v are the *endpoints* of uv. The neighborhood of a vertex v, denoted by $N_G(v)$, is the set of vertices adjacent to v and its degree $d_G(v)$ is $|N_G(v)|$. For a set $S \subseteq V(G)$, $N_G(S)$ denotes the neighborhood of S, i.e., $N_G(S) = (\bigcup_{s\in S} N_G(s)) \setminus S$. The subscripts in the above notation are omitted when the context is clear.

For a set of edges $F \subseteq E(G)$, $V(F)$ is the set of vertices that are endpoints of edges in F. For $S \subseteq V(G)$, we denote the graph obtained by deleting S from G by $G - S$, i.e., the vertex set and edge set of $G - S$ are $V(G) \setminus S$ and $\{uv \in E(G) \mid u, v \notin S\}$, respectively. Furthermore, the subgraph of G induced by S is the graph $G[S] = G \setminus (V(G) \setminus S)$. For two subsets $S_1, S_2 \subseteq V(G)$, we say S_1, S_2 are *adjacent* if there exists an edge in G with one endpoint in S_1 and the other endpoint in S_2.

A path $P_t = (v_1, v_2, \ldots, v_t)$ on t vertices, where $t \in \mathbb{N}$ is the graph with vertex set $\{v_1, v_2, \ldots, v_t\}$ and edge set $\{v_i v_{i+1} \mid i \in [t-1]\}$. Furthermore, P_t is a path between v_1 and v_t. A graph G is *connected* if for every distinct vertice $u, v \in V(G)$, there is a path (which is subgraph of G) between u and v. Consider a graph G. A (vertex inclusionwise) maximal connected subgraph of G is a *connected component* or a *component* of G. A set $A \subseteq V(G)$ is a *connected set* in G if $G[A]$ is a connected graph.

Consider a graph G and an edge $e = uv \in E(G)$. The graph obtained after "contracting" the edge e in G is denoted by G/e. That is, $V(G/e) = (V(G) \cup \{w_e\}) \setminus \{u, v\}$ and $E(G/e) = \{xy \mid x, y \in V(G) \setminus \{u, v\}, xy \in E(G)\} \cup \{w_e x \mid x \in (N_G(u) \cup N_G(v)) \setminus \{u, v\}\}$, where w_e is a newly added vertex. In the above, for an edge $ux \in E(G) \setminus \{uv\}$, the edge $w_e x \in E(G/e)$ is the *renamed* edge of ux. For $F \subseteq E(G)$, G/F denotes the graph obtained from G by contracting each (renamed) edge in F. (We note that the order in which the edges in F are contracted is immaterial.)

A graph G is *isomorphic* to a graph H if there exists a bijective function $\phi : V(G) \to V(H)$, such that for $v, u \in V(G)$, $uv \in E(G)$ if and only if $(\phi(v), \phi(u)) \in E(H)$. A graph G is *contractible* to a graph H if there exists $F \subseteq E(G)$, such that G/F is isomorphic to H. In other words, G is contractible to H if there is a surjective function $\varphi : V(G) \to V(H)$, with $W(h) = \{v \in V(G) \mid \varphi(v) = h\}$, for $h \in V(H)$, with the following properties:

- for any $h \in V(H)$, the graph $G[W(h)]$ is connected, and
- for any two vertices $h, h' \in V(H)$, $hh' \in E(H)$ if and only if $W(h)$ and $W(h')$ are adjacent in G.

Let $\mathcal{W} = \{W(h) \mid h \in V(H)\}$. The sets in $\mathcal{W}$ are called *witness sets*, and $\mathcal{W}$ is an *H-witness structure* of G.

In this paper, we will restrict ourselves to contraction to paths. This allows us to use an ordered notation for witness sets, rather than just the set notation. This ordering of the sets in witness sets is given by the ordering of vertices in the path. That is, for a $P_t = (h_1, h_2, \ldots, h_t)$-witness structure, $\mathcal{W} = \{W(h_1), W(h_2), \ldots, W(h_t)\}$ of a graph G, we use the ordered witness structure notation, $(W(h_1), W(h_2), \ldots, W(h_t))$, or simply $(W_1, W_2, \ldots, W_t)$.

In the following, we give some useful observations regarding contraction to paths.

Observation 2.1. Any connected graph can be contracted to P_2.

Observation 2.2. Let G be a graph contractible to P_t. Then, there is a P_t-witness structure, $\mathcal{W} = (W_1, \ldots, W_t)$, of G such that W_1 is a singleton set. Moreover, if $t \geq 3$, then there is a P_t-witness structure, $\mathcal{W} = (W_1, \ldots, W_t)$, of G such that both W_1 and W_t are singleton sets.

We end this section with an observation which will be used to bound the number of subsets of a set U which are of size at most $\mu|U|$ for a fixed positive constant μ which is strictly less than 1/2. We start with the following inequality for integers n and k such that $k \leq n$:

$$\binom{n}{k} \leq \left[\left(\frac{k}{n}\right)^{-\frac{k}{n}} \cdot \left(1 - \frac{k}{n}\right)^{\frac{k}{n}-1} \right]^n.$$

Using the above inequality we get the following upper bound on summand for $k < n/2$:

$$\sum_{i=1}^{k} \binom{n}{i} \leq k \cdot \binom{n}{k} \leq k \cdot \left[\left(\frac{k}{n}\right)^{-\frac{k}{n}} \cdot \left(1 - \frac{k}{n}\right)^{\frac{k}{n}-1} \right]^n.$$

For a positive constant $\mu < 1/2$, such that $k \leq \mu n$, the above inequalities can be written as

$$\sum_{i=1}^{\lfloor \mu n \rfloor} \binom{n}{i} \leq \mu n \cdot \left[\mu^{-\mu} \cdot (1-\mu)^{\mu-1} \right]^n,$$

$$\sum_{i=1}^{\lfloor \mu n \rfloor} \binom{n}{i} \leq \mu n \cdot \left[\frac{1}{\mu^{\mu}} \cdot \frac{1}{(1-\mu)^{1-\mu}} \right]^n = \mu n [g(\mu)]^n, \tag{2.1}$$

where function $g(\mu)$ is defined as

$$g(\mu) = \frac{1}{\mu^{\mu} \cdot (1-\mu)^{(1-\mu)}}.$$

The following observation is implied by the above inequalities.

Observation 2.3. For a set U with n elements and a constant $\mu < 1/2$, the number of subsets of U of size at most μn is bounded by $\mathcal{O}^\star([g(\mu)]^n)$. Moreover, all such subsets can be enumerated in time $\mathcal{O}^\star([g(\mu)]^n)$.

For a graph G, a nonempty set $Q \subseteq V(G)$, and integers $a, b \in \mathbb{N}$, a connected set A in G is a (Q, a, b)-*connected set* if $Q \subseteq A$, $|A| = a$, and $|N(A)| \leq b$. Moreover, a connected set A in G is an (a, b)-*connected set* if $|A| \leq a$ and $|N(A)| \leq b$. Next, we state results regarding (Q, a, b)-connected sets and connected sets, which follow from Lemma 3.1 of [9]. (We note that their result gives slightly better bounds, but for simplicity, we only use the bounds stated in the following lemmas.)

LEMMA 2.4. *For a graph G, a nonempty set $Q \subseteq V(G)$, and integers $a, b \in \mathbb{N}$, the number of (Q, a, b)-connected sets in G is at most $2^{a+b-|Q|}$. Moreover, we can enumerate all (Q, a, b)-connected sets in G in time $2^{a+b-|Q|} \cdot n^{\mathcal{O}(1)}$.*

LEMMA 2.5. *For a graph G and integers $a, b \in \mathbb{N}$ the number of (a, b)-connected sets in G is at most $2^{a+b} \cdot n^{\mathcal{O}(1)}$. Moreover, we can enumerate all such sets in $2^{a+b} \cdot n^{\mathcal{O}(1)}$ time.*

3. 3-Disjoint Connected Subgraph.

In this section, we define a generalization of 2-Disjoint Connected Subgraphs (2-DCS), called 3-Disjoint Connected Subgraphs (3-DCS). We design an algorithm for 3-DCS running in time $\mathcal{O}^\star(1.88^n)$, where n is number of vertices in input graph. This algorithm will be useful in designing our algorithm for Path Contraction.

In the following, we formally define the problem 2-DCS which is studied in [5, 16].

2-Disjoint Connected Subgraphs (2-DCS) **Input:** A connected graph G and two disjoint sets Z_1 and Z_2. **Question:** Does there exist a partition (V_1, V_2) of $V(G)$ such that for each $i \in [2]$, $Z_i \subseteq V_i$ and $G[V_i]$ is connected?

In the following we state a result regarding 2-DCS which will be useful in later sections.

Proposition 3.1 (see [16, Theorem 3]). *There exists an algorithm that solves the* 2-Disjoint Connected Subgraphs *problem in* $\mathcal{O}^\star(1.7804^n)$ *time where* n *is the number of vertices in the input graph.*

In the 3-DCS problem, the input is the same as that of 2-DCS, but we are interested in a partition of $V(G)$ into three sets, rather than two. We formally define the problem below.

3-Disjoint Connected Subgraphs (3-DCS) **Input:** A connected graph G and two disjoint sets Z_1 and Z_2. **Question:** Does there exist a partition (V_1, U, V_2) of $V(G)$ such that (1) for each $i \in [2]$, $Z_i \subseteq V_i$ and $G[V_i]$ is connected, (2) $G[U]$ is connected, and (3) $G - U$ has exactly two connected components, namely, $G[V_1]$ and $G[V_2]$?

We note that the problem definitions for 2-DCS and 3-DCS do not require the sets Z_1, Z_2 to be nonempty. If either set is empty, we can guess a vertex for each of the nonempty sets. Since there are at most n^2 such guesses, it will not affect the exponential factor in the running time of our algorithm. Thus, hereafter we assume that both Z_1 and Z_2 are nonempty sets.

To design our algorithm for 3-DCS, we first design an algorithm for a special case for the problem where the size of $Z_1 \cup Z_2$ is at most δn, where $\delta = 0.092$. (The choice of δ will be clear when we present the proof.) We refer to the special case of 3-DCS where $|Z_1 \cup Z_2| \leq \delta n$ as Small 3-DCS. In section 3.1, we will design an algorithm for Small 3-DCS running in time $\mathcal{O}^\star(1.88^n)$. That is, our goal of section 3.1 will be to prove the following lemma.

Lemma 3.2. Small 3-DCS *admits an algorithm running in time* $\mathcal{O}^\star(1.88^n)$*, where* n *is the number of vertices in the input graph.*

In the rest of this section, we show how we can use the above lemma to design an algorithm for the problem 3-DCS, running in time $\mathcal{O}^\star(1.88^n)$. We also show how we can obtain an algorithm for P_5-Contraction running in time $\mathcal{O}^\star(1.88^n)$, using our algorithm for 3-DCS.

In the following theorem, we give our algorithm for 3-DCS, using Lemma 3.2 as a subroutine.

Theorem 3.3. 3-DCS *admits an algorithm running in time* $\mathcal{O}^\star(1.88^n)$*, where* n *is the number of vertices in the input graph.*

Proof. Let (G, Z_1, Z_2) be an instance of 3-DCS. We consider the following two cases based on whether or not $|Z_1 \cup Z_2| \leq \delta n$, where $\delta = 0.092$. If $|Z_1 \cup Z_2| \leq \delta n$, then we resolve the instance in time $\mathcal{O}^\star(1.88^n)$, using Lemma 3.2. Now we consider the case when $|Z_1 \cup Z_2| > \delta n$. The goal is to look for a solution (V_1, U, V_2) for the instance. We begin by enumerating all potential candidates for the set U. That is, we compute the set $\mathcal{U} = \{U' \mid U' \subseteq V(G) \setminus (Z_1 \cup Z_2)\}$. As $|V(G) \setminus (Z_1 \cup Z_2)| \leq (1-\delta)n \leq 0.908n$, the time required to compute $\mathcal{U}$ is bounded by $\mathcal{O}^\star(2^{0.908n}) \in \mathcal{O}^\star(1.88^n)$. Now for each

$U' \in \mathcal{U}$, we check the following properties: (1) $G[U']$ is connected and (2) $G - U'$ has exactly two connected components, one containing all the vertices from Z_1 and the other containing all the vertices from Z_2. If there is $U' \in \mathcal{U}$ which satisfies the above two conditions, then we return that the instance is a yes-instance, and otherwise we return that the instance a no-instance. The correctness of the algorithm and the analysis of the claimed running time bound are apparent from the description. ◻

Using Theorem 3.3 we obtain our algorithm for P_5-Contraction, in the following lemma.

Lemma 3.4. *P_5-Contraction admits an algorithm running in time $\mathcal{O}^\star(1.88^n)$, where n is the number of vertices in the input graph.*

Proof. Let G be a graph. By Observation 2.2, if G is contractible to P_5, then there exists a P_5-witness structure $\mathcal{W} = (W_1, \ldots, W_5)$ of G such that W_1 and W_5 are singleton sets. We guess the pair of vertices which are in the sets W_1 and W_5, respectively. Note that there are at most $\mathcal{O}(n^2)$ choices for such pairs. Let $W_1 = \{x\}$ and $W_5 = \{y\}$ be the current guess of these sets. If there is witness structure where $W_1 = \{x\}$ and $W_5 = \{y\}$, then the vertices in $N(x)$ and $N(y)$ must belong to the sets $W(t_2)$ and $W(t_4)$, respectively. (Otherwise, the contracted graph cannot be an (induced) path on five vertices.) Note that with the above guess, the problem boils down to solving 3-DCS on the instance $(G - \{x, y\}, N_G(x), N_G(y))$. Thus, we can use Theorem 3.3 to resolve the instance $(G - \{x, y\}, N_G(x), N_G(y))$ of 3-DCS in time $\mathcal{O}^\star(1.88^n)$. This concludes the proof. ◻

3.1. Algorithm for Small 3-DCS.

The goal of this section will be to obtain a proof of Lemma 3.2, i.e., to design an algorithm for Small 3-DCS running in time $\mathcal{O}^\star(1.88^n)$. Let (G, Z_1, Z_2) be an instance of Small 3-DCS. Note that $|Z_1 \cup Z_2| \leq \delta n$, where $\delta = 0.092$.

The intuition behind our algorithm is the following. We start by showing the existence of a special type of a solution, which we call an *immovable tri-partition*, for a yes-instance. Roughly speaking, we use the properties ensured by a special solution to enumerate "connectors" for the set $Z_1 \cup Z_2$ in an auxiliary graph. To enumerate such "connectors," we employ the algorithm of Telle and Villanger [16]. Then we show how we use these potential $Z_1 \cup Z_2$ "connectors" in an auxiliary graph, to resolve the instance.

In the following, we introduce some notation and preliminary results that will be useful in designing our algorithm.

Notation and preliminary results. Consider a graph H and set $Z \subseteq V(H)$. A vertex $v \in V(H)$ is called a *Z-separator* if Z contains vertices from at least two connected components of $G - \{v\}$. A set $S \subseteq V(H)$ is a *Z-connector* if $Z \subseteq S$ and $H[S]$ is connected. Moreover, if no strict subset of S is a Z-connector, then S is a *minimal Z-connector*.

We state a result regarding enumeration of minimal Z-connectors in a graph which will be used in our algorithm.

Proposition 3.5 (see [16]). *Consider a graph H on n vertices and a set $Z \subset V(H)$ with at most $n/3$ vertices. Then, the number of minimal Z-connectors in H is bounded by $\binom{n-|Z|}{|Z|-2} \cdot 3^{(n-|Z|)/3}$. Moreover, we can enumerate all such minimal Z-connectors in time $\mathcal{O}^\star(\binom{n-|Z|}{|Z|-2} \cdot 3^{(n-|Z|)/3})$.*

In the following remark we state a criterion when we can directly conclude that the instance is a no-instance of Small 3-DCS. The correctness of this remark will easily follow from the problem definition. (Henceforth we shall assume that the premise of the remark does not hold.)

Remark 3.6. If there is an edge between Z_1 and Z_2, then conclude that (G, Z_1, Z_2) is a no-instance of SMALL 3-DCS.

A partition of $V(G)$ into three sets (V_1, U, V_2) is a *solution tri-partition* if the following conditions are satisfied: (1) for $i \in [2]$, $G[V_i]$ is connected and $Z_i \subseteq V_i$, (2) $G[U]$ is connected, and (3) $G - U$ has exactly two connected components, namely, $G[V_1]$ and $G[V_2]$.

Now we will define a "special solution", which will be called an *immovable tri-partition*, and we will show that if there is a solution, then there is also an immovable tri-partition. Our goal will be to find an immovable tri-partition, if it exists. Roughly speaking, an immovable tri-partition is a solution in which no vertex from $V_1 \cup V_2$ can be "moved" to the set U.

DEFINITION 3.7 (immovable tri-partition). A solution tri-partition (V_1, U, V_2) for (G, Z_1, Z_2) of 3-DCS is an *immovable tri-partition* if for every $i \in [2]$ and $v \in (V_i \setminus Z_i) \cap N(U)$ is a Z_i-separator in $G[V_i]$.

In the following claim we show that an immovable tri-partition exists for a yes-instance.

CLAIM 3.8. *If* (G, Z_1, Z_2) *is a yes-instance of* 3-DCS*, then there is an immovable tri-partition.*

Proof. Let (V_1, U, V_2) be a solution tri-partition of $V(G)$. If this is an immovable tri-partition, then we are done. Otherwise, assume that there is $v \in (V_1 \setminus Z_1) \cap N(U)$ such that v is not a Z_1-separator in $G[V_1]$. (The case when there is $v \in (V_2 \setminus Z_2) \cap N(U)$ such that v is not a Z_2-separator in $G[V_2]$ can be handled analogously.) Let $C_1, C_2, \ldots, C_d$ be the connected components of $G[V_1] - v$, where $d \geq 1$. Since v is not a Z_1-separator, we know that Z_1 is contained in one of the connected components. Let C_1 be the connected component which contains Z_1. Consider the tri-partition (V_1', U', V_2) of $V(G)$ where $V_1' = V(C_1) = V_1 \setminus (\{v\} \cup V(C_2) \cup \cdots \cup V(C_d))$ and $U' = U \cup \{v\} \cup C_2 \cup \cdots \cup C_d$. This tri-partition is also a solution partition as both $V_1' = C_1$ and U' are connected and V_1' contains Z_1. Following the above procedure, for a given tri-partition we can either find a vertex to move from $V_1 \cup V_2$ to U or conclude that it is an immovable tri-partition. ∎

In the following claims we establish some useful properties regarding immovable tri-partitions.

CLAIM 3.9. *Let* (G, Z_1, Z_2) *be a yes-instance of* 3-DCS *and* (V_1, U, V_2) *be an immovable tri-partition. Furthermore, let* S_1 *be a minimal* Z_1*-connector in* $G[V_1]$ *and* S_2 *be a minimal* Z_2*-connector in* $G[V_2]$*. Then, no connected component of* $G[V_1] - S_1$ *or* $G[V_2] - S_2$ *is adjacent to* U*.*

Proof. Consider a connected component C of $G[V_1] - S_1$. (The other case can be argued analogously.) As S_1 is Z_1-connector, $Z_1 \subseteq S_1$ and $G[S_1]$ is connected. Since (V_1, U, V_2) is an immovable tri-partition, no $v \in V(C)$ can be adjacent to a vertex in U, as v is not a Z_1-separator (see Definition 3.7). This concludes the proof. ∎

As was mentioned earlier, we will construct an auxiliary graph and relate a connector in the auxiliary graph to a solution for our instance. We now describe this auxiliary graph. Arbitrarily fix vertices $z_1 \in Z_1$ and $z_2 \in Z_2$. By G', we denote the graph obtained from G by adding the edge z_1z_2 to G. (From Remark 3.6 we know that there is no edge between Z_1 and Z_2 in G.) In the next claim we relate immovable tri-partitions of G with minimal connectors in G'.

CLAIM 3.10. *Let (G, Z_1, Z_2) be a yes-instance of* 3-DCS *and let (V_1, U, V_2) be an immovable tri-partition. Furthermore, let S_1 be a minimal Z_1-connector in $G[V_1]$ and S_2 be a minimal Z_2-connector in $G[V_2]$. Then, $S = S_1 \cup S_2$ is a minimal $(Z_1 \cup Z_2)$-connector in G'.*

Proof. We first argue that S is a $(Z_1 \cup Z_2)$-connector in G'. As $G'[S_1]$ and $G'[S_2]$ are connected and the edge z_1z_2 has one endpoint in S_1 and another in S_2, the graph $G'[S]$ is connected. Since S contains $Z_1 \cup Z_2$, it is a $(Z_1 \cup Z_2)$-connector.

It remains to argue that no proper subset of S is a $(Z_1 \cup Z_2)$-connector. For the sake of contradiction, suppose that there is $v \in S$ such that $S' = S \setminus \{v\}$ is a $(Z_1 \cup Z_2)$-connector in G'. We assume that $v \in S_1$. (The case when $v \in S_2$ can be argued symmetrically.) Let $S_1' = S_1 \setminus \{v\}$. Note that $G[S_1']$ is not connected, as S_1 is a minimal Z_1-connector in $G[V_1]$. Let C be a connected component of $G[S_1']$ which does not contain z_1 (which exists as $G[S_1']$ is not connected). Recall that $S_1' \subseteq V_1$, $S_2 \subseteq V_2$, and $V_1 \cap V_2 = \emptyset$. Note that there can be no edge between $V(C)$ and S_2 in G'. This contradicts that $G'[S_1' \cup S_2]$ is connected. This concludes the proof. ◻

We say that a minimal $(Z_1 \cup Z_2)$-connector S in G' is *realized* by an immovable tri-partition (V_1, U, V_2) of G if S can be partitioned into two sets, S_1, S_2, such that S_1 is a minimal Z_1-connector in $G[V_1]$ and S_2 is a minimal Z_2-connector in $G[V_2]$. Note that Claim 3.10 implies that every immovable tri-partition of $V(G)$ realizes at least one minimal $(Z_1 \cup Z_2)$-connector in G'.

The algorithm. We are now ready to design our algorithm for SMALL 3-DCS. Recall that $|Z_1 \cup Z_2| \leq \delta n = 0.092n$ and there is no edge between Z_1 and Z_2 (see Remark 3.6). Let $Z = Z_1 \cup Z_2$. Recall that G' is the graph obtained from G by adding the edge z_1z_2. Compute the set $\mathcal{S}$ of all minimal Z-connectors in G' using Proposition 3.5. (The premise of the proposition is satisfied as $|Z| \leq 0.092n \leq n/3$.) We construct a set $\mathcal{S}_{\mathsf{rel}} \subseteq \mathcal{S}$ of *relevant sets* as follows. Let $\mathcal{S}_{\mathsf{rel}} = \{S \in \mathcal{S} \mid G[S]$ have exactly two connected components $G[S_1]$ and $G[S_2]$ such that $Z_1 \subseteq S_1$ and $Z_2 \subseteq S_2\}$.

Consider $S \in \mathcal{S}_{\mathsf{rel}}$. Let $\mathcal{C}_S$ be the set of connected components in $G - S$. Let $\mathcal{C}_S^{\mathsf{bth}}$ be the set of components in $G - S$ that have a neighbor in both S_1 and S_2. That is, $\mathcal{C}_S^{\mathsf{bth}} = \{C \in \mathcal{C}_S \mid N(C) \cap S_1 \neq \emptyset \text{ and } N(C) \cap S_2 \neq \emptyset\}$.

Our algorithm does the following. If there is $S \in \mathcal{S}_{\mathsf{rel}}$, such that $|\mathcal{C}_S^{\mathsf{bth}}| = 1$, then the algorithm returns that (G, Z_1, Z_2) is a yes-instance of SMALL 3-DCS. Otherwise it returns that the instance is a no-instance.

In the following lemma we prove the correctness of the algorithm.

LEMMA 3.11. *The algorithm presented for* SMALL 3-DCS *is correct.*

Proof. In the forward direction, let (G, Z_1, Z_2) be a yes-instance of SMALL 3-DCS. We will show that there is $S \in \mathcal{S}_{\mathsf{rel}}$ such that $|\mathcal{C}_S^{\mathsf{bth}}| = 1$. Consider an immovable tri-partition (V_1, U, V_2) for the instance (its existence is guaranteed by Claim 3.8). Note that for $i \in [2]$, $Z_i \subseteq V_i$ and $G[V_i]$ is connected. Furthermore, $G[U]$ is connected and there are exactly two connected components in $G - U$, namely, $G[V_1]$ and $G[V_2]$. For $i \in [2]$, as $G[V_i]$ is connected, there is a minimal Z_i-connector S_i, in $G[V_i]$. Let $S = S_1 \cup S_2$ (see Figure 1). From Claim 3.10, S is a minimal Z-connector in G', where $Z = Z_1 \cup Z_2$. Thus, $S \in \mathcal{S}$. Note that $G[S]$ has exactly two connected components, namely, $G[S_1]$ and $G[S_2]$, and thus $S \in \mathcal{S}_{\mathsf{rel}}$. Recall $\mathcal{C}_S$ denotes the set of connected components in $G - S$ and $\mathcal{C}_S^{\mathsf{bth}}$ denotes the set of connected components in $G - S$

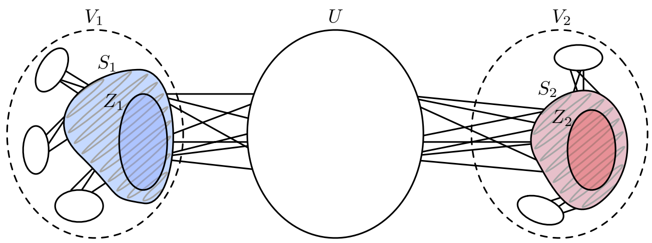

FIG. 1. *An illustration of various sets in the proof of forward direction of Lemma* 3.11.

which have neighbors both in S_1 and S_2. To obtain the proof we will show that $\mathcal{C}_S^{\mathsf{bth}} = \{G[U]\}$. Since $G[U]$ is connected and $S \cap U = \emptyset$, there is a component $C \in \mathcal{C}_S$ such that $U \subseteq V(C)$. We first show that $V(C) \setminus U = \emptyset$. Toward a contradiction, assume that $V(C) \setminus U \neq \emptyset$. Then $V(C) \cap (V_1 \cup V_2) \neq \emptyset$. Suppose that $V(C) \cap V_1 \neq \emptyset$. (The case when $V(C) \cap V_2 \neq \emptyset$ can be argued analogously.) Then there is a vertex $v \in V(C) \cap V_1$ such that $v \in (V_1 \setminus Z_1) \cap N(U)$. From the above we can contradict the fact that (V_1, U, V_2) is an immovable-tripartition. Thus we conclude that $V(C) = U$. Note that U is adjacent to both V_1 and V_2 in G and $G[U]$ is a connected component of $G - S$. Hence, $\emptyset \subset N(U) \cap V_1 \subseteq S_1$ and $\emptyset \subset N(U) \cap V_2 \subseteq S_2$. Thus, $G[U] \in \mathcal{C}_S^{\mathsf{bth}}$. As no vertex in $V_1 \cup V_2$ can be adjacent to both S_1 and S_2, we conclude that $\{G[U]\} = \mathcal{C}_S^{\mathsf{bth}}$.

In the reverse direction, assume that there is $S \in \mathcal{S}_{\mathsf{rel}}$ such that $|\mathcal{C}_S^{\mathsf{bth}}| = 1$. We will construct a solution (V_1, U, V_2) for the instance (G, Z_1, Z_2) and hence establish that the instance is a yes-instance of SMALL 3-DCS. Let C^* be the unique connected component in $\mathcal{C}_S^{\mathsf{bth}}$. As $S \in \mathcal{S}_{\mathsf{rel}}$, $G[S]$ has exactly two connected components, C_1 and C_2, such that $Z_1 \subseteq V(C_1)$ and $Z_2 \subseteq V(C_2)$. For $i \in [2]$, let $\mathcal{C}_i$ be the set of connected components different from C^* that have a neighbor in S_i. Note that $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \{C^*\} = \mathcal{C}_S$ (and they are pairwise disjoint). Let $V_1 = V(C_1) \cup (\bigcup_{C \in \mathcal{C}_1} V(C))$, $V_2 = V(C_2) \cup (\bigcup_{C \in \mathcal{C}_2} V(C))$, and $U = V(C^*)$. Note that (V_1, U, V_2) is a partition of $V(G)$, for each $i \in [2]$, $G[V_i]$ is connected and $Z_i \subseteq V_i$, $G[U]$ is connected, and $G - U$ has exactly two connected components, namely, $G[V_1]$ and $G[V_2]$. (In the above we rely on the connectedness of G.) Hence, (G, Z_1, Z_2) is a yes-instance of SMALL 3-DCS. ∎

Proof of Lemma 3.2. From Lemma 3.11, we know that the algorithm presented for SMALL 3-DCS is correct. Thus to establish the proof of the lemma, it is enough to argue that the algorithm presented for SMALL 3-DCS runs in time $\mathcal{O}^\star(1.88^n)$. The only step of the algorithm that requires exponential time is the computation of set $\mathcal{S}$ of all minimal Z-connectors in G' where we use Proposition 3.5. As $|Z| = |Z_1 \cup Z_2| \leq \delta n = 0.092n$, the time required to compute $\mathcal{S}$ is bounded by $\mathcal{O}^\star(\binom{n-|Z_1 \cup Z_2|}{|Z_1 \cup Z_2|-2} \cdot 3^{(n-|Z_1 \cup Z_2|)/3})$, which is bounded by $\mathcal{O}^\star(\binom{(1-\delta)n}{\delta n} \cdot 3^{(1-\delta)n/3})$. Using Observation 2.3, we can bound the above by $\mathcal{O}^\star((\frac{(1-\delta)^{(1-\delta)}}{\delta^\delta \cdot (1-2\delta)^{(1-2\delta)}})^n)$. Using a computer program we obtained that $\delta \sim 0.092$ would be the value for which the overall running time of 3-DCS is optimized. Thus, the running time of our algorithm is bounded by $\mathcal{O}^\star(1.88^n)$. ∎

4. Exact algorithm for PATH CONTRACTION. In this section we design our algorithm for PATH CONTRACTION running in time $\mathcal{O}^\star(1.99987^n)$, where n is the number of vertices in the input graph. To design our algorithm, we design four different subroutines each solving the problem PATH CONTRACTION. Each of these subroutines is better than the other when a specific "type" of solution exists for the input instance. Thus the main algorithm will use these subroutines to search for solutions of the type they are the best for. We also design a subroutine for enumerating special types of partial solutions, which will be used in some of our algorithms for Path Contraction.

In the following we briefly explain the four subroutines and describe when they are useful. Let G be an instance for Path Contraction, where G is a graph on n vertices. Let t be the largest integer (which we do not know a priori), such that G is contractible to P_t with $(W_1, W_2, \ldots, W_t)$ as a P_t-witness structure of G. We let $\mathtt{OS}$ and $\mathtt{ES}$ be the union of vertices in odd and even witness sets, respectively. That is, $\mathtt{OS} = \bigcup_{x=1}^{\lceil t/2 \rceil} W_{2x-1}$ and $\mathtt{ES} = \bigcup_{x=1}^{\lfloor t/2 \rfloor} W_{2x}$.

We now give an intuitive idea of the purposes of each of our subroutines in the main algorithm, while deferring their implementations to the subsequent sections. We also describe a subroutine which will help us build "partial solutions," and this subroutine will be used in two of our subroutines for Path Contraction. (We refer the reader to Figure 2 for an illustration of it.)

Balanced PC. This subroutine is useful when we can "break" the graph into two parts after a witness set, such that the closed neighborhood for each of the parts has small size, or in other words, the parts are "balanced." The quantification of the "balancedness" after a witness set will be done with the help of a rational number $0 < \alpha \leq 1$, which will be part of the input for the subroutine. The subroutine will only look for those P_t-witness structures for G for which there is an integer $i \in [t]$, such that the sizes of both $N[\bigcup_{j \in [i]} W_j]$ and $N[\bigcup_{j \in [t] \setminus [i]} W_j]$ are bounded by αn. Moreover, the algorithm will return the largest such t. Our algorithm for Balanced PC will run in time $\mathcal{O}^\star(2^{\alpha n})$. Note that when $\alpha = 1$, Balanced PC is an algorithm for Path Contraction running in time $\mathcal{O}^\star(2^n)$.

2-Union Heavy PC. This subroutine will be used when a "large" part of the graph is concentrated in two consecutive witness sets and the neighborhood of the rest of the graph into them is "small." The quantification of terms "large/small" will

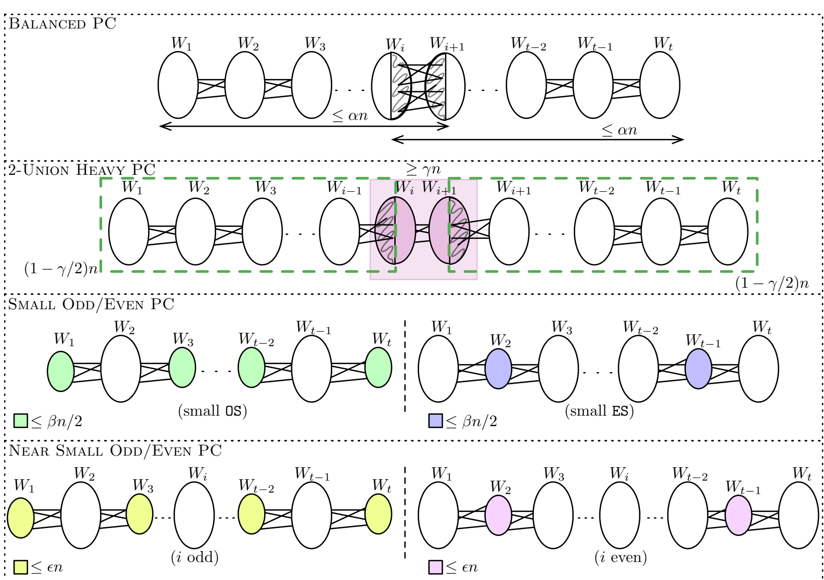

Fig. 2. *Various subroutines for the algorithm and their usage.*

be done by a fraction $0 < \gamma < 1$, which will be part of the input. The algorithm will only search for those P_t-witness structure of G where there is an integer $i \in [t-1]$, such that $|W_i \cup W_{i+1}| \geq \gamma n$, and $|N[\bigcup_{j\in[i-1]} W_j]|, |N[\bigcup_{j\in[t]\setminus[i+1]} W_j]| \leq (1-\gamma/2)n$. Moreover, the algorithm will return the largest such t.

Small Odd/Even PC. Roughly speaking, this subroutine is particularly useful when one of OS or ES is "small." The "smallness" of OS/ES is quantified by a rational number $0 < \beta \leq 1$, which will be part of the input. The subroutine will only look for those P_t-witness structures for G where one of $|\mathtt{OS}| \leq \beta n/2$ or $|\mathtt{ES}| \leq \beta n/2$ holds. Moreover, the algorithm will return the largest integer $t \geq 1$, for which such a P_t-witness structure for G exists. Small Odd/Even PC will run in time $\mathcal{O}^\star(c^n)$, where $c = g(\beta/2)$. We note that when $\beta = 1$, then one of $|\mathtt{OS}| \leq \beta n/2$ or $|\mathtt{ES}| \leq \beta n/2$ definitely holds. Thus, for $\beta = 1$, Small Odd/Even PC is an algorithm for Path Contraction running in time $\mathcal{O}^\star(2^n)$ (see Observation 2.3).

Near Small Odd/Even PC. In the case when both OS and ES are "large," it may be the case that for one of OS/ES, there is just one witness set which is large. That is, when we remove this large witness set, then one of OS/ES becomes "small." The "smallness" of the remaining OS/ES (after removing a witness set) will be quantified by a rational number $0 < \epsilon \leq 1$, which will be part of the input. The subroutine will only look for those P_t-witness structures for G where the size of one of $|\mathtt{OS}|$ or $|\mathtt{ES}|$ after removal of a witness set is bounded by ϵn. Moreover, the algorithm will return the largest such t.

Our subroutines Balanced PC and 2-Union Heavy PC use a subroutine called Enum-Partial-PC for enumerating solutions for "small" subgraphs. The efficiency of the algorithm for Enum-Partial-PC is centered around the bounds for (Q, a, b)-connected sets. In section 4.1 we (define and) design an algorithm for Enum-Partial-PC. In sections 4.2, 4.3, 4.4, and 4.5 we present our algorithms for Balanced PC, 2-Union Heavy PC, Small Odd/Even PC, and Near Small Odd/Even PC, respectively. Finally, in section 4.6 we show how we can use the above algorithms to obtain an algorithm for Path Contraction, running in time $\mathcal{O}^\star(1.99987^n)$.

4.1. Algorithm for Enum-Partial-PC. In this section, we describe an algorithm which computes a "nice solution" for all "ρ-small" subsets of vertices of an input graph. In an input graph G, for a set $S \subseteq V(G)$, by $\Phi(S)$ we denote the set of vertices in S that have a neighbor outside S. That is, $\Phi(S) = \{s \in S \mid N(s) \setminus S \neq \emptyset\}$. A set $S \subseteq V(G)$ is *ρ-small* if $N[S] \leq \rho n$. For a ρ-small set $S \subseteq V(G)$, the largest integer t_S is called the *nice solution* if $G[S]$ is contractible to P_{t_S} with all the vertices in $\Phi(S)$ in one of the end bags. That is, there is a P_{t_S}-witness structure $(W_1, W_2, \ldots, W_{t_S})$ of $G[S]$, such that $\Phi(S) \subseteq W_{t_S}$. We formally define the problem Enum-Partial-PC in the following way.

Enum-Partial-PC
Input: A graph G on n vertices and a fraction $0 < \rho \leq 1$.
Output: A table Γ which is indexed by *ρ-small* sets. For any *ρ-small* set S, $\Gamma[S]$ is the largest integer t for which $G[S]$ has a P_t-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_t)$, such that $\Phi(S) \subseteq W_t$.

We design an algorithm for Enum-Partial-PC running in time $\mathcal{O}^\star(2^{\rho n})$. We briefly explain how we can compute nice solutions for the ρ-small set. Consider a ρ-small set S. Note that $|S| \leq \rho n$. Thus, by the method of 2-coloring (as was explained in the introduction), we can obtain the nice solution in time $2^{\rho n}$. This would lead us to an algorithm running in time $\mathcal{O}^\star(2^{\rho n} g(\rho)^n)$ (see inequality (2.1)). By doing a simple

Procedure 4.1 Algorithm for Enum-Partial-PC

Input: A graph G and a fraction $0 < \rho \leq 1$.
Output: A table Γ such that for every ρ-small set S, $\Gamma[S]$ is the largest integer q for which $G[S]$ can be contracted to P_q with $\Phi(S)$ is in the end bag.

Compute $\mathcal{S} = \{S \subseteq V(G) \mid G[S]$ is connected and $|N[S]| \leq \rho n\}$ (Lemma 2.5)
for $S \in \mathcal{S}$ **do**
  Initialize $\Gamma[S] = 1$
end for
for $S \in \mathcal{S}$ (in increasing order of their sizes) **do**
  for every pair (a, b) of positive integers s.t. $|S| + a + b \leq \rho n$ and $|N(S)| \leq a$ **do**
    Compute $\mathcal{A}_{a,b}[S] = \{A \subseteq V(G - S) \mid G - S[A]$ is connected, $N_G(S) \subseteq A, |A| = a,$ and $|N_{G-S}(A)| = b\}$, using Lemma 2.4
    for $A \in \mathcal{A}_{a,b}[S]$ **do**
      $\Gamma[S \cup A] = \max\{\Gamma[S \cup A], \Gamma[S] + 1\}$
    end for
  end for
end for
return Γ

dynamic programming we can also obtain an algorithm running in time $\mathcal{O}^\star(3^n)$. We will improve upon these algorithms by a dynamic programming algorithm where we update the values "forward" instead of looking "backward."

The algorithm. We start by defining the table entries for our dynamic programming routine, which is used for computation of nice solutions. (The pseudocode for our algorithm is presented in Procedure 4.1.) Let $\mathcal{S}$ be the set of connected ρ-small sets. That is, $\mathcal{S} = \{S \subseteq V(G) \mid G[S]$ is connected and $|N[S]| \leq \rho n\}$. For each $S \in \mathcal{S}$, there is an entry, denoted by $\Gamma[S]$, in the table which stores a nice solution for S. In other words, $\Gamma[S]$ is the largest integer $q \geq 1$ for which $G[S]$ can be contracted to P_q with a P_q-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_q)$ of $G[S]$, such that $\Phi(S) \subseteq W_q$. The algorithm starts by initializing $\Gamma[S] = 1$ for each $S \in \mathcal{S}$.

In the following we introduce some notation that will be useful in stating the algorithm. Consider $S \in \mathcal{S}$. We will define a set $\mathcal{A}[S]$, which will be the set of all "potential extenders bags" for S, when we look at contraction to paths for larger graphs (containing S). For the sake of notational simplicity, we will define $\mathcal{A}_{a,b}[S] \subseteq \mathcal{A}[S]$, where the sets in $\mathcal{A}_{a,b}[S]$ will be of size exactly a and will have exactly b neighbors outside S. We will define the above sets only for "relevant" as and bs. We now move to the formal description of these sets. Consider $S \in \mathcal{S}$ and integers a, b, such that $|S| + a + b \leq \rho n$ and $|N(S)| \leq b$. We let $\mathcal{A}_{a,b}[S] = \{A \subseteq V(G - S) \mid G - S[A]$ is connected, $N_G(S) \subseteq A, |A| = a,$ and $|N_{G-S}(A)| = b\}$.

The algorithm now computes nice solutions. The algorithm considers sets from $S \in \mathcal{S}$, in increasing order of their sizes, and does the following. (Two sets that have the same size can be considered in any order.) For every pair of integers a, b, such that $|S| + a + b \leq \rho n$ and $|N(S)| \leq b$, it computes the set $\mathcal{A}_{a,b}[S]$. Note that $\mathcal{A}_{a,b}[S]$ can be computed in time $\mathcal{O}^\star(2^{a+b-|S|})$, using Lemma 2.4. Now the algorithm considers $A \in \mathcal{A}_{a,b}[S]$. Intuitively speaking, A is the "new" witness set to be "appended" to the witness structure of $G[S]$, to obtain a witness structure for $G[S \cup A]$. Thus, the algorithm sets $\Gamma[S \cup A] = \max\{\Gamma[S \cup A], \Gamma[S] + 1\}$. This finishes the description of our algorithm.

In the following few lemmas we establish the correctness and runtime analysis of the algorithm.

LEMMA 4.1. *For each $S \in \mathcal{S}$, the algorithm computes $\Gamma[S]$ correctly.*

Proof. We prove the statement by induction on the size of sets in $\mathcal{S}$. The base case is for sets of size 1. That is, for the base case we show that for each $S \in \mathcal{S}$, such that $|S| = 1$, the algorithm computes $\Gamma[S]$ correctly. Consider a set $S \in \mathcal{S}$ of size 1. Note that in this case, $\Gamma[S]$ must be equal to 1. At step 2, the algorithm initializes $\Gamma[S'] = 1$ for each $S' \in \mathcal{S}$. Note that no other step of the algorithm modifies the value of $\Gamma[S]$ (as $|S| = 1$). Thus, the algorithm correctly computes $\Gamma[S]$.

For the induction hypothesis, we assume that the algorithm computes $\Gamma[S']$ correctly, for each $S' \in \mathcal{S}$, such that $|S'| \leq r$. We will now argue that the computation of $\Gamma[\cdot]$ for sets of size $r + 1$ is correct. Consider $S \in \mathcal{S}$ such that $|S| = r + 1$. Let q_{opt} be the nice solution for S and q_{out} be the value of $\Gamma[S]$ computed by the algorithm. We will show that $q_{\mathsf{out}} = q_{\mathsf{opt}}$.

First, we show that $q_{\mathsf{out}} \geq q_{\mathsf{opt}}$. Consider a $P_{q_{\mathsf{opt}}}$-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_{q_{\mathsf{opt}}})$ of $G[S]$ such that $\Phi(S) \subseteq W_{q_{\mathsf{opt}}}$. Note that $q_{\mathsf{out}} \geq 1$; thus if $q_{\mathsf{opt}} = 1$, then $q_{\mathsf{out}} \geq q_{\mathsf{opt}}$ trivially holds. Now we consider the case when $q_{\mathsf{opt}} \geq 2$. Let $\widehat{S} = S \setminus W_{q_{\mathsf{opt}}}$, $a = |W_{q_{\mathsf{opt}}}|$, and $b = |N(W_{q_{\mathsf{opt}}})|$. As $S \in \mathcal{S}$, we have $|N[S]| \leq \rho n$. Thus, $|\widehat{S}| + a + b \leq \rho n$. Since $\mathcal{W}$ is a $P_{q_{\mathsf{opt}}}$-witness structure of $G[S]$, we have $N(\widehat{S}) \subseteq W_{q_{\mathsf{opt}}}$ and thus $a \geq |N(\widehat{S})|$. Also, $\emptyset \neq \widehat{S} \in \mathcal{W}$. (In the above we rely on the fact that $q_{\mathsf{opt}} \geq 2$.) From the above discussions we can conclude that the set $\mathcal{A}_{a,b}[\widehat{S}]$ is well defined, and $W_{q_{\mathsf{opt}}} \in \mathcal{A}_{a,b}[\widehat{S}]$. By the induction hypothesis, we known that $\Gamma[\widehat{S}] \geq q_{\mathsf{opt}} - 1$ is correctly computed. Thus, at step 9, the algorithm sets $q_{\mathsf{out}} = \Gamma[S] \geq q_{\mathsf{opt}}$. Hence, we conclude that $q_{\mathsf{out}} \geq q_{\mathsf{opt}}$.

Next, we show that $q_{\mathsf{out}} \leq q_{\mathsf{opt}}$. Note that $q_{\mathsf{opt}} \geq 1$. Thus, if $q_{\mathsf{out}} = 1$, then the claim is trivially satisfied. We next consider the case when $q_{\mathsf{out}} \geq 2$. As $q_{\mathsf{out}} \geq 2$, there is a set $\widehat{S} \in \mathcal{S}$, with $\widehat{S} \subset S$ ($\widehat{S} \neq S$) and integers a, b, such that $|\widehat{S}| + a + b \leq \rho n$, $|N(\widehat{S})| \leq a$, such that $A = S \setminus \widehat{S} \in \mathcal{A}_{a,b}[\widehat{S}]$ and $q_{\mathsf{out}} = \Gamma[\widehat{S}] + 1$. By the induction hypothesis, $\Gamma[\widehat{S}]$ is computed correctly. Thus, there is a $P_{q_{\mathsf{out}}-1}$-witness structure $\mathcal{W}' = (W_1, W_2, \ldots, W_{q_{\mathsf{out}}-1})$ of $G[\widehat{S}]$ such that $\Phi(\widehat{S}) \subseteq W_{q_{\mathsf{out}}-1}$. But then, $\mathcal{W} = (W_1, W_2, \ldots, W_{q_{\mathsf{out}}-1}, A)$ is a P_{out}-witness structure of $G[S]$. Thus, $q_{\mathsf{out}} \leq q_{\mathsf{opt}}$. This concludes the proof. ∎

LEMMA 4.2. *The algorithm presented for* ENUM-PARTIAL-PC *runs in time $\mathcal{O}^\star(2^{\rho n})$.*

Proof. Steps 1 and 2 of the algorithm can be executed in time bounded by $\mathcal{O}^\star(2^{\rho n})$ (see Lemma 2.5). We will now argue about the time required for execution of the *for*-loop starting at step 5. Toward this, we start by partitioning sets in $\mathcal{S}$ by their sizes and the sizes of their neighborhood. Recall that for any $S \in \mathcal{S}$, we have $|N[S]| \leq \rho n$. Let $\mathcal{S}_{x,y} = \{S \in \mathcal{S} \mid |S| = x \text{ and } |N(S)| = y\}$, where $x, y \in [\lfloor \rho n \rfloor]$, such that $x + y \leq \rho n$. Consider $x, y \in [\lfloor \rho n \rfloor]$, where $x + y \leq \rho n$. From Lemma 2.5, $|\mathcal{S}_{x,y}|$ is bounded by $\mathcal{O}^\star(2^{x+y})$. For each $S \in \mathcal{S}_{x,y}$, the algorithm considers every pair of integers a, b, such that $|S| + a + b \leq \rho n$ and $|N(S)| \leq a$, and computes the set $\mathcal{A}_{a,b}[S]$. Note that $\mathcal{A}_{a,b}[S]$ can be computed in time bounded by $\mathcal{O}^\star(2^{a+b-|N(S)|})$, from Lemma 2.4. Furthermore, the algorithm spends time proportional to $|\mathcal{A}_{a,b}[S]|$ at step 9. From the above discussions, we can bound the running time of the algorithm by the following:

$$\mathcal{O}^\star\left(\sum_{\substack{x,y\\x+y\le\rho n}}\ \sum_{\substack{a,b\\x+a+b\le\rho n}} 2^{x+y}\cdot 2^{a+b-y}\right)=\mathcal{O}^\star\left(\sum_{\substack{x,y\\x+y\le\rho n}}\ \sum_{\substack{a,b\\x+a+b\le\rho n}} 2^{x+a+b}\right)=\mathcal{O}^\star(2^{\rho n})$$

This concludes the proof. ∎

4.2. Algorithm for Balanced PC. We formally define the problem Balanced PC in the following.

Balanced PC
Input: A graph G on n vertices and a fraction $0 < \alpha \le 1$.
Output: Largest integer $t \ge 2$ for which G has a P_t-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_t)$ such that there is $i \in [t]$ with $N[\bigcup_{j\in[i]} W_j] \le \alpha n$ and $N[\bigcup_{j\in[t]\setminus[i]} W_j] \le \alpha n$. Moreover, if no such t exists, then output 1.

We design an algorithm for Balanced PC running in time $\mathcal{O}^\star(2^{\alpha n})$. Let (G,α) be an instance of Balanced PC.

We begin by explaining the intuition behind the algorithm. Recall that for an α-small set $S \subseteq V(G)$, integer t_S is called the *nice solution* if $G[S]$ is contractible to P_{t_S} with all the vertices in $\Phi(S)$ in the end bag. That is, there is a P_{t_S}-witness structure $(W_1, W_2, \ldots, W_{t_S})$ of $G[S]$ such that $\Phi(S) \subseteq W_{t_S}$. Suppose that we know the value of t_S for every α-small set S. Now we see how we can use these nice solutions for α-small sets to solve our problem (see Figure 3). Recall that we are looking for the largest integer t, such that G is contractible to P_t, with $\mathcal{W} = (W_1, W_2, \ldots, W_t)$ as a P_t-witness structure of G, such that there is $i \in [t]$ with $|\bigcup_{j\in[i+1]} W_j| \le \alpha n$ and $|\bigcup_{j\in[t]\setminus[i-1]} W_j| \le \alpha n$. Let $S = \bigcup_{j\in[i]} W_j$. As $|\bigcup_{j\in[i+1]} W_j| \le \alpha n$ and $N(S) \subseteq W_{i+1}$, the set S is an α-small set. Similarly, we can argue that $V(G)\setminus S$ is an α-small set. Thus, for S and $V(G)\setminus S$, we know the nice solutions t_S and $t_{V(G)\setminus S}$, respectively. Notice that the solution to the whole graph is actually $t_S + t_{V(G)\setminus S}$.

The algorithm. The algorithm initializes $t = 1$. (At the end, t will be the output of the algorithm.) The algorithm computes table $\Gamma =$Enum-Partial-PC(G,α) using Procedure 4.1. Let $\mathcal{S}$ be the set of all connected sets S in G such that $|N[S]| \le \alpha n$. That is, $\mathcal{S} = \{S \subseteq V(G) \mid G[S]$ is connected and $|N[S]| \le \alpha n\}$. For each $S \in \mathcal{S}$, we have an entry denoted by $\Gamma[S]$. The algorithm considers each $S \in \mathcal{S}$ for which $V(G)\setminus S \in \mathcal{S}$. It sets $t = \max\{t, \Gamma[S] + \Gamma[V(G)\setminus S]\}$. Finally, the algorithm returns t as the output. This completes the description of the algorithm.

Lemma 4.3. *The algorithm presented for* Balanced PC *is correct.*

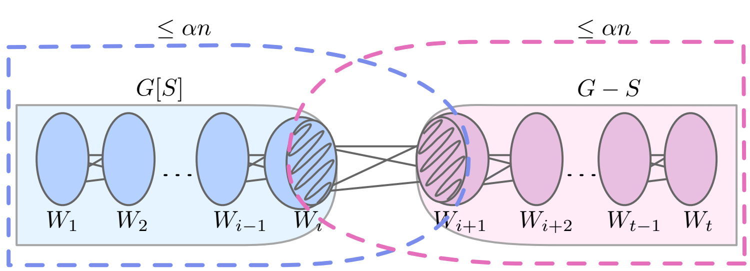

Fig. 3. *An illustration of construction of the solution using solutions for instances of smaller sizes.*

Proof. For an instance (G, α), suppose that t_{opt} is the solution to Balanced PC and t_{out} is the output returned by the algorithm. We will show that $t_{\mathsf{out}} = t_{\mathsf{opt}}$.

First, we show that $t_{\mathsf{out}} \geq t_{\mathsf{opt}}$. As $t_{\mathsf{out}} \geq 2$, if $t_{\mathsf{opt}} = 2$, then the claim trivially holds. Thus, we assume that $t_{\mathsf{opt}} \geq 3$. Let $\mathcal{W} = (W_1, W_2, \ldots, W_{t_{\mathsf{opt}}})$ be a $P_{t_{\mathsf{opt}}}$-witness structure of G, such that there is $i \in [t_{\mathsf{opt}}]$ with $|N[\bigcup_{j\in[i]} W_j]| \leq \alpha n$ and $|N[\bigcup_{j\in[t_{\mathsf{opt}}]\setminus[i]} W_j]| \leq \alpha n$. Let $S = \bigcup_{j\in[i]} W_j$ and $\overline{S} = \bigcup_{j\in[t_{\mathsf{opt}}]\setminus[i]} W_j$. Note that $\overline{S} = V(G) \setminus S$ and $S, \overline{S} \in \mathcal{S}$. Let $\mathcal{W}_1 = (W_1, W_2, \ldots, W_i)$ and $\mathcal{W}_2 = (W_{i+1}, W_{i+2}, \ldots, W_{t_{\mathsf{opt}}})$. Note that $\mathcal{W}_1$ is a P_i-witness structure of $G[S]$ such that $\Phi(S) \subseteq W_i$. Similarly, $\mathcal{W}_2$ is a $P_{t_{\mathsf{opt}}-i}$-witness structure of $G[\overline{S}]$ such that $\Phi(\overline{S}) \subseteq W_{i+1}$. From Lemma 4.1, we know that the algorithm has correctly computed the values $\Gamma[S]$ and $\Gamma[\overline{S}]$. From the above discussions we can conclude that $\Gamma[S] \geq i$ and $\Gamma[\overline{S}] \geq t_{\mathsf{opt}} - i$. Thus, we can conclude that $t_{\mathsf{out}} \geq \Gamma[S] + \Gamma[\overline{S}] \geq t_{\mathsf{opt}}$.

Next, we show that $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$. As $t_{\mathsf{opt}} \geq 2$, if $t_{\mathsf{out}} = 2$, the condition $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$ is trivially satisfied. Now we consider the case when $t_{\mathsf{out}} = 3$. In this case, there is a set $S \in \mathcal{S}$ such that $V(G) \setminus S \in \mathcal{S}$ and $t = \Gamma[S] + \Gamma[V(G) \setminus S]$. From Lemma 4.1, the algorithm has correctly computed $q_1 = \Gamma[S]$ and $q_2 = \Gamma[V(G) \setminus S]$. Thus, there is a P_{q_1}-witness structure $\mathcal{W}_1 = (W_1, W_2, \ldots, W_{q_1})$ for $G[S]$ such that $\Phi(S) \subseteq W_{q_1}$. Similarly, there is a P_{q_2}-witness structure $\mathcal{W}_2 = (W'_1, W'_2, \ldots, W'_{q_2})$ for $G[V(G) \setminus S]$ such that $\Phi(V(G) \setminus S) \subseteq W'_{q_2}$. Recall that $t = q_1 + q_2$. We will show that $\mathcal{W} = (W_1, W_2, \ldots, W_{q_1}, W'_{q_2}, \ldots, W'_2, W'_1)$ is a P_t-witness structure of G such that there is $i \in [t]$ with $|N[\bigcup_{j\in[i]} W_j]| \leq \alpha n$ and $|N[\bigcup_{j\in[t]\setminus[i]} W_j]| \leq \alpha n$. Lemma 4.1 and connectedness of G implies that $\mathcal{W}$ is a P_t-witness structure of G. Moreover, as $S, V(G) \setminus S \in \mathcal{S}$, for $i = q_1$, we have $|N[\bigcup_{j\in[i]} W_j]| \leq \alpha n$ and $|N[\bigcup_{j\in[t]\setminus[i]} W_j]| \leq \alpha n$. Hence, we can conclude that $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$. ∎

Lemma 4.4. *The algorithm presented for* Balanced PC *runs in time* $\mathcal{O}^\star(2^{\alpha n})$.

Proof. The algorithm for Balanced PC calls Procedure 4.1 on input (G, α) and iterates over all the values in Γ. Hence, Lemma 4.2 implies that the running time of the algorithm presented for Balanced PC is $\mathcal{O}^\star(2^{\alpha n})$. ∎

4.3. Algorithm for 2-Union Heavy PC.

We formally define the problem 2-Union Heavy PC in the following (also see Figure 4).

> 2-Union Heavy PC
> **Input:** A graph G on n vertices and a fraction $0 < \gamma \leq 1$.
> **Output:** Largest integer $t \geq 3$ for which G has a P_t-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_t)$ such that there is $i \in [t-1]$ for which the following conditions hold: (1) $|W_i \cup W_{i+1}| \geq \gamma n$ and (2) $|N[\bigcup_{j\in[i-1]} W_j]|, |N[\bigcup_{j\in[t]\setminus[i+1]} W_j]| \leq (1 - \gamma/2)n$. Moreover, if no such t exists, then output 2.

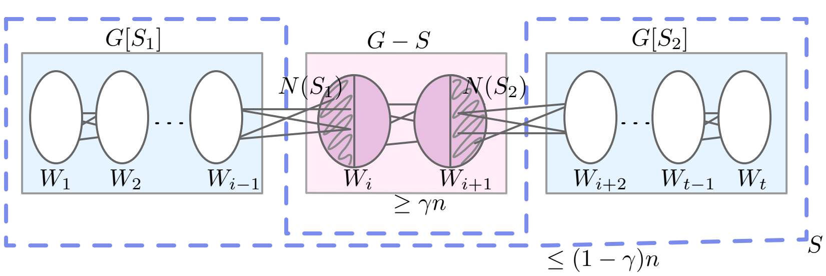

Fig. 4. *An intuitive illustration of the algorithm for* 2-Union Heavy PC.

Procedure 4.2 Algorithm for 2-Union Heavy PC

Input: A graph G and a fraction $0 < \gamma \leq 1$.
Output: An integer t.

```
Initialize t = 2
Let 𝒮 = {S ⊆ V(G) | |S| ≤ (1 − γ)n and G[S] has exactly two connected
  components G[S_1], G[S_2], s.t. |N[S_1]|, |N[S_2]| ≤ (1 − γ/2)n}
Let Ŝ = {Ŝ ⊆ V(G) | |N[Ŝ]| ≤ (1 − γ/2)n and G[Ŝ] is connected}. Compute
  the value of Γ[S], for each S ∈ Ŝ, by computing the table Γ = Enum-Partial-
  PC(G, 1 − γ/2)
for S ∈ 𝒮 do
  Let S_1 and S_2 be the two connected components of G[S]
  if (G − S, N_G(S_1), N_G(S_2)) is a yes instance of 2-DCS then
    t = max{t, Γ[S_1] + Γ[S_2] + 2}
  end if
end for
return t
```

We design an algorithm for 2-Union Heavy PC running in time $\mathcal{O}^\star(2^{(1-\gamma/2)n} + c^n)$, where $c = \max_{\gamma \leq \delta \leq 1}\{1.7804^\delta \cdot g(1-\delta)\}$. The first term in the running time expression will be due to a call made to Enum-Partial-PC with $\rho = (1-\gamma/2)$, and the second term will be due to enumerating sets of size at most $(1-\gamma)n$ and running the algorithm for solving 2-Disjoint Connected Subgraphs for an instance created for each of them, using the algorithm of Telle and Villanger [16].

We now formally describe our algorithm. The algorithm will output an integer t, which is initially set to 2. Let $\mathcal{S} = \{S \subseteq V(G) \mid |S| \leq (1-\gamma)n$ and $G[S]$ has exactly two connected components $G[S_1], G[S_2],$ such that $|N[S_1]|, |N[S_2]| \leq (1-\gamma/2)n\}$. Let $\widehat{\mathcal{S}} = \{\widehat{S} \subseteq V(G) \mid |N[\widehat{S}]| \leq (1-\gamma/2)n$ and $G[\widehat{S}]$ is connected$\}$. The algorithm will now compute a table Γ, which has an entry $\Gamma[\widehat{S}]$, for each $\widehat{S} \in \widehat{\mathcal{S}}$. The definition of Γ is the same as that in section 4.2, where $\rho = 1-\gamma/2$. That is, for $\widehat{S} \in \widehat{\mathcal{S}}$, $\Gamma[\widehat{S}]$ is the largest integer $q \geq 1$ for which $G[\widehat{S}]$ can be contracted to P_q with a P_q-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_q)$ of $G[\widehat{S}]$, such that $\Phi(\widehat{S}) \subseteq W_q$. Compute the value of $\Gamma[\widehat{S}]$, for each $\widehat{S} \in \widehat{\mathcal{S}}$, by using Enum-Partial-PC$(G, 1-\gamma/2)$. For each $S \in \mathcal{S}$, the algorithm does the following. Recall that $G[S]$ has exactly two connected components. Let the two connected components in $G[S]$ be $G[S_1]$ and $G[S_2]$, where $S_1 \cup S_2 = S$. Recall that $|N[S_1]|, |N[S_2]| \leq (1-\gamma/2)n$. Thus, $S_1, S_2 \in \widehat{\mathcal{S}}$. If $(G - S, N_G(S_1), N_G(S_2))$ is a yes-instance of 2-DCS, then the algorithm sets $t = \max\{t, \Gamma[S_1] + \Gamma[S_2] + 2\}$, and otherwise, it moves to the next set in $\widehat{\mathcal{S}}$. Finally, the algorithm outputs t. This completes the description of the algorithm. See Procedure 4.2.

In the following two lemmas we present the correctness and runtime analysis of the algorithm, respectively.

Lemma 4.5. *The algorithm presented for* 2-Union Heavy PC *is correct.*

Proof. For an instance (G, γ), suppose that t_{opt} is the solution to 2-Union Heavy PC and t_{out} is the output returned by the algorithm. We will show that $t_{\mathsf{out}} = t_{\mathsf{opt}}$.

First, we show that $t_{\mathsf{out}} \geq t_{\mathsf{opt}}$. As $t_{\mathsf{out}} \geq 2$, if $t_{\mathsf{opt}} = 2$, then the claim trivially holds. Thus, we assume that $t_{\mathsf{opt}} \geq 3$. Let $\mathcal{W} = (W_1, W_2, \ldots, W_{t_{\mathsf{opt}}})$ be a $P_{t_{\mathsf{opt}}}$-witness structure of G such that there is $i \in [t_{\mathsf{opt}} - 1]$ for which the following conditions hold: (1) $|W_i \cup W_{i+1}| \geq \gamma n$ and (2) $|N[\bigcup_{j\in[i-1]} W_j]|, |N[\bigcup_{j\in[t_{\mathsf{opt}}]\setminus[i+1]} W_j]| \leq (1-\gamma/2)n$. Let $Z = W_i \cup W_{i+1}$ and $S = V(G) \setminus Z$. As $|Z| \geq \gamma n$, we have $|S| \leq (1-\gamma)n$. Also, as $\mathcal{W}$ is a $P_{t_{\mathsf{opt}}}$-witness structure of G, $G[S]$ has exactly two connected components $G[S_1]$ and $G[S_2]$, where $S_1 = \bigcup_{j\in[i-1]} W_j$ and $S_2 = \bigcup_{j\in[t_{\mathsf{opt}}]\setminus[i+1]} W_j$. Note that we have $|N[S_1]| \leq (1-\gamma/2)n$ and $|N[S_2]| \leq (1-\gamma/2)n$. From the above discussions, we can conclude that $S \in \mathcal{S}$ and $S_1, S_2 \in \widehat{\mathcal{S}}$. By Lemma 4.1, the values of $\Gamma[S_1]$ and $\Gamma[S_2]$ are computed correctly, and we can conclude that $\Gamma[S_1]+\Gamma[S_2] \geq t_{\mathsf{opt}}-2$. Also, (W_i, W_{i+1}) is a solution to the instance $(G-S, N_G(S_1), N_G(S_2))$, and hence it is a yes-instance of 2-DCS. Thus, $t_{\mathsf{out}} = t \geq \Gamma[S_1] + \Gamma[S_2] + 2 \geq t_{\mathsf{opt}}$.

Next, we show that $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$. As $t_{\mathsf{opt}} \geq 2$, if $t_{\mathsf{out}} = 2$, the condition $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$ is trivially satisfied. Now we consider the case when $t_{\mathsf{out}} \geq 3$. In this case, there is a set $S \in \mathcal{S}$ such that $(G-S, N_G(S_1), N_G(S_2))$ is a yes-instance of 2-DCS and $t_{\mathsf{out}} = \Gamma[S_1] + \Gamma[S_2] + 2$, where $G[S_1]$ and $G[S_2]$ are the two connected component of $G[S]$. From Lemma 4.1 it follows that the algorithm has correctly computed $q_1 = \Gamma[S_1]$ and $q_2 = \Gamma[S_2]$. Thus, there is a P_{q_1}-witness structure $\mathcal{W}_1 = (W_1, W_2, \ldots, W_{q_1})$ for $G[S_1]$, such that $N(S_1) \subseteq W_{q_1}$. Similarly, there is a P_{q_2}-witness structure $\mathcal{W}_2 = (W'_1, W'_2, \ldots, W'_{q_2})$ for $G[S_2]$ such that $N(S_2) \subseteq W'_{q_2}$. Note that $q_1 + q_2 \leq t_{\mathsf{opt}} - 2$. Let (Z_1, Z_2) be a solution to 2-DCS in $(G-S, N_G(S_1), N_G(S_2))$. Note that $\mathcal{W} = (W_1, W_2, \ldots, W_{q_1}, Z_1, Z_2, W'_{q_2}, \ldots, W'_2, W'_1)$ $= (W_1, W_2, \ldots, W_{q_1}, W_{q_1+1}, W_{q_1+2}, \ldots, W_{q_1+q_2+2})$ is a $P_{t_{\mathsf{out}}}$-witness structure of G such that (1) $|W_{q_1+1} \cup W_{q_1+2}| \geq \gamma n$ and (2) $|N[\bigcup_{j\in[q_1]} W_j]|, |N[\bigcup_{j\in[t_{\mathsf{out}}]\setminus[q_1+2]} W_j]| \leq (1-\gamma/2)n$. Thus, we can conclude that $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$. ∎

LEMMA 4.6. *The algorithm presented for* 2-UNION HEAVY PC *runs in time* $\mathcal{O}^\star(2^{(1-\gamma/2)n} + c^n)$, *where* $c = \max_{\gamma\leq\delta\leq 1}\{1.7804^\delta \cdot g(1-\delta)\}$.

Proof. Using Observation 2.3, step 2 of the algorithm can be executed in time $\mathcal{O}^\star((g(1-\gamma))^n)$, which is bounded by $\mathcal{O}^\star(c^n)$. Step 3 of the algorithm can be executed in time $\mathcal{O}^\star(2^{(1-\gamma/2)n})$, by Lemmas 4.1 and 4.2. We now argue about the time required for the *for*-loop starting at step 4 (all the remaining steps can be executed in constant time). The number of sets in $\mathcal{S}$ of size at most $(1-\delta)n$ is bounded by $(g(1-\delta))^n$. For each $\gamma \leq \delta \leq 1$, and each set $S \in \mathcal{S}$ of size at most $(1-\delta)n$, we resolve the instance $(G-S, N_G(S_1), N_G(S_2))$ of 2-DCS, where $G[S_1]$ and $G[S_2]$ are the two connected components of $G-S$. Note that the number of vertices in $G-S$ is bounded by δn, and hence using Proposition 3.1, we can resolve the instance $(G-S, N_G(S_1), N_G(S_2))$ of 2-DCS in time $\mathcal{O}^\star(1.7804^{\delta n})$. From the above discussions we can conclude that the running time of the algorithm is bounded by $\mathcal{O}^\star(2^{(1-\gamma/2)n} + c^n)$, where $c = \max_{\gamma\leq\delta\leq 1}\{1.7804^\delta \cdot g(1-\delta)\}$. ∎

4.4. Algorithm for SMALL ODD /EVEN PC. We formally define the problem SMALL ODD/EVEN PC in the following.

SMALL ODD/EVEN PC
Input: A graph G on n vertices and a fraction $0 < \beta \leq 1$.
Output: Largest integer t for which G can be contracted to P_t, with $\mathcal{W} = (W_1, W_2, \ldots, W_t)$ as a P_t-witness structure of G, such that $\lvert\mathtt{OS}_{\mathcal{W}}\rvert \leq \beta n/2$ or $\lvert\mathtt{ES}_{\mathcal{W}}\rvert \leq \beta n/2$, where $\mathtt{OS}_{\mathcal{W}} = \bigcup_{i\in[\lceil t/2\rceil]} W_{2i-1}$ and $\mathtt{ES}_{\mathcal{W}} = \bigcup_{i\in[\lfloor t/2\rfloor]} W_{2i}$.

In this section, we design an algorithm for SMALL ODD/EVEN PC running in time $\mathcal{O}^\star(c^n)$, where $c = g(\beta/2)$.

Let (G, β) be an instance of SMALL ODD/EVEN PC. The algorithm is fairly simple. It starts by enumerating all "potential candidates" for $\mathtt{OS}_{\mathcal{W}}$ (resp., $\mathtt{ES}_{\mathcal{W}}$), i.e., the set of all subsets of $V(G)$ of size at most $\beta n/2$. Then, for each such "potential set," it contracts G appropriately and finds the length of the path to which G is contracted (and stores 0, if the contracted graph is not a path). Finally, it returns the maximum over such path lengths.

We now move to a formal description of the algorithm. We start by enumerating the set of all subsets of $V(G)$ of size at most $\beta n/2$. That is, $\mathcal{S} = \{S \subseteq V(G) \mid |S| \leq \beta n/2\}$. Note that $\mathcal{S}$ can be computed in time $\mathcal{O}^\star(g(\beta/2)^n)$, using Observation 2.3. For each $S \in \mathcal{S}$ the algorithm does the following. Let $\mathcal{C}_S$ and $\overline{\mathcal{C}}_S$ be the set of connected components of $G[S]$ and $G - S$, respectively. Let G_S be the graph obtained from G by contracting each $C \in \mathcal{C}_S \cup \overline{\mathcal{C}}_S$ to a single vertex. Set $\mathsf{len}_S = |V(G_S)|$ if G_S is a path and $\mathsf{len}_S = 0$ otherwise. Finally, the algorithm returns $\max_{S \in \mathcal{S}} \mathsf{len}_S$.

In the following lemma we prove the correctness and runtime analysis of the algorithm.

LEMMA 4.7. *The algorithm presented for* SMALL ODD/EVEN PC *is correct and runs in time* $\mathcal{O}^\star(g(\beta/2)^n)$.

Proof. Clearly, the algorithm presented for SMALL ODD/EVEN PC runs in time $\mathcal{O}^\star(g(\beta/2)^n)$. Now we prove the correctness of the algorithm.

In the forward direction, assume that G is contractible to $P_{t_{opt}}$, where $(W_1, W_2, \ldots, W_{t_{opt}})$ is a $P_{t_{opt}}$-witness structure of G, such that $|\mathtt{OS}_{\mathcal{W}}| \leq \beta n/2$ or $|\mathtt{ES}_{\mathcal{W}}| \leq \beta n/2$, where $\mathtt{OS}_{\mathcal{W}} = \bigcup_{i \in [\lceil t_{opt} \rceil]} W_{2i-1}$ and $\mathtt{ES}_{\mathcal{W}} = \bigcup_{i \in [\lfloor t_{opt} \rfloor]} W_{2i}$. We will show that the algorithm outputs $t_{out} \geq t_{opt}$. We assume that $|\mathtt{OS}_{\mathcal{W}}| \leq \beta n/2$. (The case when $|\mathtt{ES}_{\mathcal{W}}| \leq \beta n/2$ can be argued analogously.) Let $S = \mathtt{OS}_{\mathcal{W}}$. Note that $S \in \mathcal{S}$. The set of connected components in $G[S]$ is precisely $\mathcal{C}_S = \{G[W_{2i-1}] \mid i \in [\lceil t_{out}/2 \rceil]\}$. Also, the set of connected components in $G - S$ is precisely $\overline{\mathcal{C}}_S = \{G[W_{2i+1}] \mid i \in [\lfloor t_{out}/2 \rfloor]\}$. Thus, G_S is isomorphic to $P_{t_{opt}}$. Thus, the output of the algorithm $t_{out} = \max_{S' \in \mathcal{S}} \mathsf{len}_{S'} \geq t_{opt}$, as $S \in \mathcal{S}$ and $\mathsf{len}_S = t_{opt}$.

For the other direction, let t_{out} be the output of the algorithm. Note that $t_{out} \geq 1$, as $\emptyset \in \mathcal{S}$ and $G_\emptyset$ is a single vertex (as G is connected). Consider $S \in \mathcal{S}$ such that $\mathsf{len}_S = t_{out}$. Note that a P_t-witness set for G is $\mathcal{W} = \{V(C) \mid C \in \mathcal{C}_S\} \cup \{V(C) \mid C \in \overline{\mathcal{C}}_S\}$. Thus one of $\mathtt{OS}_{\mathcal{W}} = S$ or $\mathtt{ES}_{\mathcal{W}} = S$ must hold. Moreover, as $S \in \mathcal{S}$, we have $|S| \leq \beta n/2$. This concludes the proof. ∎

4.5. Algorithm for NEAR SMALL ODD /EVEN PC. We formally define the problem NEAR SMALL ODD/EVEN PC in the following (also see Figure 5).

NEAR SMALL ODD/EVEN PC
Input: A graph G on n vertices and a fraction $0 < \epsilon \leq 1$.
Output: Largest integer $t \geq 3$ for which there is a P_t-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_t)$ of G, for which there is $i \in \{2, 3, \ldots, t-1\}$, such that if i is odd, then $|\mathtt{OS}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$ and otherwise, $|\mathtt{ES}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$. Here, $\mathtt{OS}_{\mathcal{W}} = \bigcup_{i \in [\lceil t/2 \rceil]} W_{2i-1}$ and $\mathtt{ES}_{\mathcal{W}} = \bigcup_{i \in [\lfloor t/2 \rfloor]} W_{2i}$. If no such $t \geq 3$ exists, then output 2.

We design an algorithm for NEAR SMALL ODD/EVEN PC running in time $\mathcal{O}^\star(c^n)$ where $c = \max_{0 \leq \delta \leq \epsilon}\{1.88^{(1-\delta)} \cdot g(\delta)\}$. The second term in the multiplicative factor will be due to enumeration of sets, and the first term will be due to calls made to the algorithm for 3-DISJOINT CONNECTED SUBGRAPHS, from section 3.

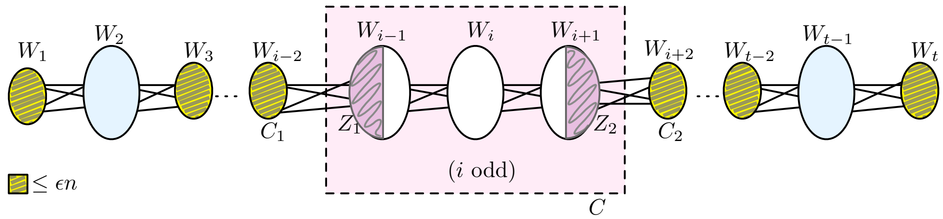

FIG. 5. *An intuitive illustration of the algorithm for* NEAR SMALL ODD/EVEN PC.

Let (G, ϵ) be an instance of NEAR SMALL ODD/EVEN PC. We start by explaining the intuitive idea behind our algorithm (see Figure 5). Consider a P_t-witness structure $\mathcal{W} = (W_1, W_2, \ldots, W_t)$ of G, for which there is $i \in \{2, 3, \ldots, t-2\}$, such that if i is odd, then $|\mathtt{OS}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$ and otherwise, $|\mathtt{ES}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$. In the above, $\mathtt{OS}_{\mathcal{W}} = \bigcup_{i \in [\lceil t/2 \rceil]} W_{2i-1}$ and $\mathtt{ES}_{\mathcal{W}} = \bigcup_{i \in [\lfloor t/2 \rfloor]} W_{2i}$. Let us consider the case when i is odd (the other case is symmetric). Let $S = \mathtt{OS}_{\mathcal{W}} \setminus W_i$. (The union of vertices from yellow sets in Figure 5 is the set S.) As $|S| \leq \epsilon n$, the algorithm starts by enumerating all "potential candidates" for the set S. All the components of $G - S$, except for the component C, containing W_i, must each be contracted to a single vertex. Similarly, each connected components of $G[S]$ must be contracted to a single vertex. Moreover, the component containing W_i must be "split" into three sets. The first and the last sets in the "split" must contain the neighbors of W_{i-2} and W_{i+2} in C, respectively. To obtain such a "split," we use the algorithm for 3-DISJOINT CONNECTED SUBGRAPHS that we designed in section 3.

We now formally describe our algorithm. (A pseudocode of our algorithm is presented in Procedure 4.3.) The algorithm will output an integer t, which is initially set to 2. Let $\mathcal{S} = \{S \subset V(G) \mid |S| \leq \epsilon n\}$. For each $S \in \mathcal{S}$, the algorithm does the following. Let $\mathcal{C}_S$ and $\overline{\mathcal{C}}_S$ be the sets of connected components in $G[S]$ and $G - S$, respectively. Let H_S be obtained from G by contracting component in $\mathcal{C}_S \cup \overline{\mathcal{C}}_S$ to single vertices. That is, H_S has a vertex v_C for each $C \in \mathcal{C}_S \cup \overline{\mathcal{C}}_S$, and two vertices $v_C, v_{C'} \in V(H_S)$ are adjacent in H_S if and only if C and C' are adjacent in G. If H_S is not a path, then the algorithm moves to the next set in $\mathcal{S}$. Otherwise, for each $C^* \in \overline{\mathcal{C}}_S$ it does the following. Intuitively speaking, C^* is the current guess for the component containing vertices from W_i for the witness structure that we are looking for. Note that C^* can be adjacent to at most two components from $\mathcal{C}_S$, as H_S is a path. Moreover, C^* must be adjacent to at least one component from $\mathcal{C}_S$, as G is connected and S is a strict subset of $V(G)$, i.e., $S \neq V(G)$. Let C_1 be a component from $\mathcal{C}_S$ that is adjacent to C^* in G, and $Z_1 = N(C_1) \cap V(C^*)$. Let $C_2 \in \mathcal{C}_S \setminus \{C_1\}$ be a component of $G[S]$ that is adjacent to C^*, and $Z_2 = N(C_2) \cap V(C^*)$. If such a C_2 does not exist, then we set $Z_2 = \emptyset$. If $(G[C^*], Z_1, Z_2)$ is a yes-instance of 3-DCS, then the algorithm updates $t = \max\{t, |V(H_S)| + 2\}$. After finishing the processing for each $S \in \mathcal{S}$, the algorithm outputs t. This finishes the description of our algorithm.

In the following two lemmas we present the correctness and runtime analysis of the algorithm, respectively.

LEMMA 4.8. *The algorithm presented for* NEAR SMALL ODD/EVEN PC *is correct.*

Proof. For an instance (G, ϵ), suppose that t_{opt} is the solution to NEAR SMALL ODD/EVEN PC and t_{out} is the output returned by the algorithm. We will show that $t_{\mathsf{out}} = t_{\mathsf{opt}}$.

Procedure 4.3 Algorithm for Near Small Odd/Even PC.

Input: A graph G and a fraction $0 < \epsilon \leq 1$.
Output: An integer t.

Initialize $t = 2$
Let $\mathcal{S} = \{S \subset V(G) \mid |S| \leq \epsilon n\}$
for $S \in \mathcal{S}$ **do**
  Let $\mathcal{C}_S$ and $\overline{\mathcal{C}}_S$ be the sets of connected components in $G[S]$ and $G - S$, resp.
  Let H_S be obtained from G by contracting components in $\mathcal{C}_S \cup \overline{\mathcal{C}}_S$ to single vertices
  if H_S is a path **then**
    for $C^* \in \overline{\mathcal{C}}_S$ **do**
      Let $C_1 \in \mathcal{C}_S$ be a component of $G[S]$ that is adjacent to C^*, and $Z_1 = N(C_1) \cap V(C^*)$
      Let $C_2 \in \mathcal{C}_S \setminus \{C_1\}$ be a component of $G[S]$ that is adjacent to C^*, and $Z_2 = N(C_2) \cap V(C^*)$, if such a C_2 does not exist, then $Z_2 = \emptyset$
      if $(G[C^*], Z_1, Z_2)$ is a yes-instance of 3-DCS **then**
        $t = \max\{t, |V(H_S)| + 2\}$
      end if
    end for
  end if
end for
return t

First, we show that $t_{\mathsf{out}} \geq t_{\mathsf{opt}}$. As $t_{\mathsf{out}} \geq 2$, if $t_{\mathsf{opt}} = 2$, then the claim trivially holds. Thus, we assume that $t_{\mathsf{opt}} \geq 3$. Let $\mathcal{W} = (W_1, W_2, \dots, W_{t_{\mathsf{opt}}})$ be a $P_{t_{\mathsf{opt}}}$-witness structure of G, for which there is $i \in \{2, 3, \dots, t_{\mathsf{opt}} - 1\}$, such that if i is odd, then $|\mathtt{OS}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$, and otherwise, $|\mathtt{ES}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$. In the above, $\mathtt{OS}_{\mathcal{W}} = \bigcup_{i \in [\lceil t_{\mathsf{opt}}/2 \rceil]} W_{2i-1}$ and $\mathtt{ES}_{\mathcal{W}} = \bigcup_{i \in [\lfloor t_{\mathsf{opt}}/2 \rfloor]} W_{2i}$. We consider the case when i is odd. (The case when i is even can be argued analogously.) Let $S = \mathtt{OS}_{\mathcal{W}} \setminus W_i$. As $|S| \leq \epsilon n$, we have $S \in \mathcal{S}$. Note that H_S is a path. Let $C^* \in \overline{\mathcal{C}}_S$ be the connected component of $G - S$ containing W_i. Let $C_1 \in \mathcal{C}_S$ be a connected component of $G[S]$ adjacent to C^*, and $Z_1 = N(C_1) \cap V(C^*)$. Consider $C_2 \in \mathcal{C}_S \setminus \{C_1\}$ that is adjacent to C^*, and let $Z_2 = N(C_2) \cap V(C^*)$. If such a C_2 does not exist, then set $Z_2 = \emptyset$. Note that $(G[C^*], Z_1, Z_2)$ is a yes-instance of 3-DCS, as (W_{i-1}, W_i, W_{i+1}) is a solution to it. In the above we rely on the fact that $i \in \{2, 3, \dots, t-1\}$, and thus each of W_{i-1} and W_{i+1} is nonempty. From the above discussions we can conclude that $t_{\mathsf{out}} \geq t_{\mathsf{opt}} = V(H_S) + 2$ (as C^* is split into three witness sets).

Next, we show that $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$. As $t_{\mathsf{opt}} \geq 2$, if $t_{\mathsf{out}} = 2$, the condition $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$ is trivially satisfied. Now we consider the case when $t_{\mathsf{out}} \geq 3$. There is $S \in \mathcal{S}$ for which H_S is a path and there is $C^* \in \overline{\mathcal{C}}_S$, for which the instance $(G[C^*], Z_1, Z_2)$ is a yes-instance of 3-DCS. Let (V_1^*, U^*, V_2^*) be a solution to 3-DCS for the instance $(G[C^*], Z_1, Z_2)$. Let $\mathcal{W}' = \mathcal{C}_S \cup (\overline{\mathcal{C}}_S \setminus \{C^*\}) \cup \{V_1^*, U^*, V_2^*\}$. Note that $|\mathcal{W}'| = |V(H_S)| + 2$ and $\mathcal{W}'$ is a $P_{t_{\mathsf{out}}}$-witness structure for G. Let $\mathcal{W} = (W_1, W_2, \dots, W_{t_{\mathsf{out}}})$ be the ordered witness structure corresponding to the $P_{t_{\mathsf{out}}}$-witness structure $\mathcal{W}'$ of G. Note that there is $i \in \{2, 3, \dots, t_{\mathsf{out}} - 1\}$ such that $V(C^*) \subseteq W_{i-1} \cup W_i \cup W_{i+1}$. Thus we can conclude that $\mathcal{W}$ is a $P_{t_{\mathsf{out}}}$-witness structure of G, for which there is $i \in \{2, 3, \dots, t_{\mathsf{out}} - 1\}$, such that if i is odd, then $|\mathtt{OS}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$, and otherwise, $|\mathtt{ES}_{\mathcal{W}} \setminus W_i| \leq \epsilon n$. From the above discussions we can conclude that $t_{\mathsf{out}} \leq t_{\mathsf{opt}}$. ∎

LEMMA 4.9. *The algorithm presented for* NEAR SMALL ODD/EVEN PC *runs in time* $\mathcal{O}^\star(c^n)$*, where* $c = \max_{0\le\delta\le\epsilon}\{1.88^{(1-\delta)} \cdot g(\delta)\}$.

Proof. From Observation 2.3, step 2 of the algorithm can be executed in time $\mathcal{O}^\star([g(\epsilon)]^n)$. Also, $|\mathcal{S}|$ is bounded by $\mathcal{O}^\star([g(\epsilon)]^n)$. For a set $S \in \mathcal{S}$, step 10 can be executed in time $\mathcal{O}^\star(1.88^{n-|S|})$, from Theorem 3.3, and other steps can be executed in polynomial time. Hence, the running time algorithm can be bounded by $\mathcal{O}^\star(c^n)$, where $c = \max_{0\le\delta\le\epsilon}\left\{1.88^{(1-\delta)} \cdot g(\delta)\right\}$. ∎

4.6. Algorithm for PATH CONTRACTION. We are now ready to present our algorithm for PATH CONTRACTION. The algorithm calls four of the subroutines SMALL ODD/EVEN PC, BALANCED PC, 2-UNION HEAVY PC, and NEAR SMALL ODD/EVEN PC for appropriate instances, and returns the maximum of their outputs. In the following theorem, we present the algorithm, which is the main result of this paper.

THEOREM 4.10. PATH CONTRACTION *admits an algorithm running in time* $\mathcal{O}^\star(1.99987^n)$*, where* n *is the number of vertices in the input graph.*

Proof. We fix α, β, γ such that they satisfy following inequalities: (1) $2 - \alpha - \beta/2 + \gamma/2 \le \alpha$; (2) $1 - \gamma/2 \le \alpha$. These inequalities will be used in the later parts of the proof. We set $\alpha = 0.9996$, $\beta = 0.9885$, $\gamma = 0.9864$, and $\epsilon = 1 - \beta/2 - \gamma/2$.

The algorithm for PATH CONTRACTION is as follows. Let G be the input graph. Let t_1 = SMALL ODD/EVEN PC(G, β), t_2 =BALANCED PC(G, α), t_3 =2-UNION HEAVY PC(G, γ), and t_4 = NEAR SMALL ODD/EVEN PC(G, ϵ). Furthermore, let $t^* = \max\{2, t_1, t_2, t_3, t_4\}$. The algorithm returns t^*. This finishes the description of the algorithm.

By Lemma 4.7, t_1 can be computed in time $\mathcal{O}^\star(1.99987^n)$. By Lemma 4.4, t_2 can be computed in time $\mathcal{O}^\star(1.9995^n)$. From Lemma 4.6, t_3 can be computed in time $\mathcal{O}^\star(1.9133^n)$. From Lemma 4.9, t_4 can be computed in time $\mathcal{O}^\star(1.9953^n)$. Thus, the running time of the algorithm is bounded by $\mathcal{O}^\star(1.99987^n)$.

We now prove the correctness of the algorithm. If the algorithm returns an integer t, then from Lemmas 4.7, 4.3, 4.5, and 4.8, it follows that G is contractible to P_t. Now we prove the other direction. Suppose G is contractible to P_t. We will show that $t^* \ge t$.

Let $\mathcal{W} = (W_1, W_2, \ldots, W_t)$ be a P_t-witness structure of G. We assume that $t \ge 3$, as otherwise, trivially, $t^* \ge t$ is satisfied. We also assume that $|W_1| = |W_t| = 1$ (see Observation 2.2). Let $\texttt{OS} = \bigcup_{i\in[\lceil t/2\rceil]} W_{2i-1}$ and $\texttt{ES} = \bigcup_{i\in[\lfloor t/2\rfloor]} W_{2i}$. For $i \in [t]$, we let $Q_i = \bigcup_{j\in[i]} W_j$ and $R_i = \bigcup_{j\in[t]\setminus[i-1]} W_j$. Note that $N(Q_i)$ (resp., $N(R_i)$) is contained in $W_{i+1} \subseteq Q_{i+1}$ (resp., $W_{i-1} \subseteq N(R_{i-1})$). We use the above observation frequently in the remainder of the proof.

We say that $\mathcal{W}$ admits an α-bi-partition, if there is $j \in [t]$, such that $|\bigcup_{i\in[j]} W_i| \le \alpha n$ and $|\bigcup_{i\in[t]\setminus[j-2]} W_i| \le \alpha n$. Note that if $\mathcal{W}$ is an α-*bi-partition*, then $|N[\bigcup_{i\in[j-1]} W_i]| \le \alpha n$ and $|N[\bigcup_{i\in[t]\setminus[j-1]} W_i]| \le \alpha n$. If we show that $\mathcal{W}$ is an α-bi-partition, then using Lemma 4.3 we can conclude that $t^* \ge t$. We will use the above in later parts of our proof.

If $|\texttt{OS}| \le \beta n/2$ or $|\texttt{ES}| \le \beta n/2$, then SMALL ODD/EVEN PC is better than the other. Hence, Lemma 4.7 implies that $t^* \ge t$. Hereafter we assume that $|\texttt{OS}| > \beta n/2$ and $|\texttt{ES}| > \beta n/2$. The above implies that $\beta n/2 < |\texttt{OS}|, |\texttt{ES}| < (1 - \beta/2)n$. Note that there can be at most two witness sets in $\mathcal{W}$ which are of size more than $\gamma n/2$, as $\gamma = 0.9864$. Next, we consider cases based on the number of witness sets of size more than $\gamma n/2$ in $\mathcal{W}$.

Case 1. All witness sets in $\mathcal{W}$ are of size at most $\gamma n/2$. In this case, we will show that $\mathcal{W}$ admits an α-bi-parition. In this case the BALANCED PC subroutine is better than the other. Hence, using Lemma 4.3, we can conclude that $t^* \geq t$. Let j be the largest integer such that $|Q_j| \leq \alpha n$. Note that $j \geq 2$, as $|W_1| + |W_2| \leq \gamma n \leq \alpha n$. The above also implies that $j < t$, as $\alpha = 0.9996$. As $|Q_{j+1}| > \alpha n$, we have $|Q_j| + |W_{j+1}| > \alpha n$, which can be rewritten as $|Q_j| > \alpha n - |W_{j+1}|$. Note that (Q_j, R_{j+1}) is a partition of $V(G)$, and thus, $|Q_j| + |R_{j+1}| = n$. Hence,

$$|R_{j+1}| = n - |Q_j| < n - \alpha n + |W_{j+1}|.$$

We use this to obtain an upper bound on $|R_{j-1}|$. By definition,

$$|R_{j-1}| = |W_{j-1}| + |W_j| + |R_{j+1}|, \text{ and hence}$$

$$|R_{j-1}| < |W_{j-1}| + |W_j| + n - \alpha n + |W_{j+1}| = n - \alpha n + |W_{j-1}| + |W_j| + |W_{j+1}|.$$

Since $j-1, j+1$ have the same parity (both are odd or both are even), $|W_{j-1}| + |W_{j+1}| < (1-\beta/2)n$. From the premise of the case we have $|W_j| \leq \gamma n/2$. From the above discussions and using ineqality (2.1), we get

$$|R_{j-1}| < n - \alpha n + (1-\beta/2)n + \gamma/2n = (2 - \alpha - \beta/2 + \gamma/2)n \leq \alpha n.$$

Thus, $|\bigcup_{i\in[j]} W_i| \leq \alpha n$ and $|\bigcup_{i\in[t]\setminus[j-2]} W_i|| \leq \alpha n$. Thus, $\mathcal{W}$ admits an α-bi-parition.

Case 2. There is exactly one witness set W_k in $\mathcal{W}$ such that $|W_k| \geq \gamma n/2$. If $\mathcal{W}$ admits an α-bi-partition, then we can conclude that $t^* \geq t$, using Lemma 4.3. Thus, we assume that $\mathcal{W}$ does not admit an α-bi-partition. Let j be the largest integer, such that $|Q_j| \leq \alpha n$. (Recall that all graphs under consideration have at least two vertices, $|W_1| = 1$, and hence j exists.) As argued previously, we have $|W_{j-1}| + |W_{j+1}| < (1-\beta/2)n$. If $j \neq k$, then $|W_j| \leq \gamma n/2$, and arguments are similar to that of the previous case. We now consider a case when $j = k$. Without loss of generality, assume that k is odd. Since $|\texttt{OS}| < (1-\beta/2)n$ and $\gamma n/2 \leq |W_j|$, we have $|\texttt{OS} \setminus W_j| \leq (1-\beta/2-\gamma/2)n = \epsilon n$. In this case the SMALL ODD/EVEN PC subroutine is better than the other. Thus, from Lemma 4.8 we can conclude that $t^* \geq t$.

Case 3. There are exactly two witness sets W_j, W_k in $\mathcal{W}$, such that $|W_j|, |W_k| \geq \gamma n/2$ and $j < k$. Consider the case when $k = j+1$. Note that in the above case, we have $|N[\bigcup_{i\in[j-1]} W_i]|, |N[\bigcup_{i\in[t]\setminus[j+1]} W_i]| \leq (1-\gamma/2)n$. Thus, from Lemma 4.5 we can conclude that $t^* \geq t$. Now we consider the case when $j < k$ and $k \neq j+1$. We now consider the case when j is odd and k is even. (The case when j is even and k is odd can be argued analogously.) Note that $|\texttt{ES} \setminus W_j| \leq (1-\beta/2)n - \gamma n/2 = \epsilon n$. In this case, the 2-UNION HEAVY PC subsubroutine is better than the other. Thus, from Lemma 4.8 we can conclude that $t^* \geq t$.

Now we consider the case when j, k are both even or both odd and $k \neq j+1$. Note that in the above case $k \geq j+3$. We will conclude that $t^* \geq t$ by showing that $\mathcal{W}$ admits an α-bi-partition (and Lemma 4.3). To this end, we start by arguing that $|Q_{j+2}|, |R_{j+1}| \leq \alpha n$. As $k \geq j+3$, set $W_k \cap Q_{j+2} = \emptyset$. Thus, $|Q_{j+2}| \leq n - \gamma n/2 \leq \alpha n$. By similar arguments we can obtain that $|R_{j+1}| \leq \alpha n$. Note that the above implies that $\mathcal{W}$ admits an α-bi-partitioned witness structure. This concludes the proof. ∎

5. Conclusion. We generalized the 2-DISJOINT CONNECTED SUBGRAPHS problem to a problem called 3-DISJOINT CONNECTED SUBGRAPHS, where instead of partitioning the vertex set into two connected sets, we are required to partition it into three connected sets. We gave an algorithm for 3-DISJOINT CONNECTED SUBGRAPHS running in time $\mathcal{O}^\star(1.88^n)$. We believe that this algorithm can be of independent interest and may find other algorithmic applications. We designed an algorithm for Path Contraction which breaks the $\mathcal{O}^\star(2^n)$ barrier. It was surprising that even for a simple problem like Path Contraction, there was no known algorithm that solves it faster than $\mathcal{O}^\star(2^n)$. Our algorithm for Path Contraction relied on the fact that the number of (Q, a, b)-connected sets can be bounded by $\mathcal{O}^\star(2^{a+b-|Q|})$. This gives us savings in the number of states that we consider in our dynamic programming routine (for enumerating partial solutions). We designed four different algorithms for Path Contraction and used them for appropriate instances, to obtain the main algorithm for Path Contraction.

It is interesting to identify other graph contraction problems for which we can improve upon brute force algorithms. The simple algorithm described in section 1 can be used to solve the Tree Contraction problem. We believe there is an algorithm that breaks $\mathcal{O}^\star(2^n)$ for Tree Contraction. On the other hand, we conjecture that a brute force algorithm, running in time $\mathcal{O}^\star(n^n)$, is optimal for Clique Contraction under ETH.

REFERENCES

[1] T. Asano and T. Hirata, *Edge-contraction problems*, J. Comput. System Sci., 26 (1983), pp. 197–208.

[2] A. Björklund, *Determinant sums for undirected hamiltonicity*, SIAM J. Comput., 43 (2014), pp. 280–299.

[3] A. E. Brouwer and H. J. Veldman, *Contractibility and NP-completeness*, J. Graph Theory, 11 (1987), pp. 71–79.

[4] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström, *On problems as hard as CNF-SAT*, ACM Trans. Algorithms, 12 (2016), pp. 41:1–41:24.

[5] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk, *Solving the 2-disjoint connected subgraphs problem faster than* 2^n, Algorithmica, 70 (2014), pp. 195–207.

[6] K. K. Dabrowski and D. Paulusma, *Contracting bipartite graphs to paths and cycles*, Inform. Process. Lett., 127 (2017), pp. 37–42.

[7] R. Diestel, *Graph Theory*, 4th ed., Grad. Texts in Math. 173, Springer, New York, 2012.

[8] J. Fiala, M. Kamiński, and D. Paulusma, *A note on contracting claw-free graphs*, Discrete Math. Theor. Comput. Sci., 15 (2013), pp. 223–232.

[9] F. V. Fomin and Y. Villanger, *Treewidth computation and extremal combinatorics*, Combinatorica, 32 (2012), pp. 289–308.

[10] P. Heggernes, P. van 't Hof, B. Lévêque, and C. Paul, *Contracting chordal graphs and bipartite graphs to paths and trees*, Discrete Appl. Math., 164 (2014), pp. 444–449.

[11] R. Impagliazzo and R. Paturi, *On the complexity of k-SAT*, J. Comput. System Sci., 62 (2001), pp. 367–375.

[12] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity?*, J. Comput. System Sci., 63 (2001), pp. 512–530.

[13] W. Kern and D. Paulusma, *Contracting to a Longest Path in H-Free Graphs*, preprint, https://arxiv.org/abs/1810.01542, 2018.

[14] D. Lokshtanov, D. Marx, and S. Saurabh, *Known algorithms on graphs of bounded treewidth are probably optimal*, ACM Trans. Algorithms, 14 (2018), pp. 13:1–13:30.

[15] R. E. Tarjan and A. E. Trojanowski, *Finding a maximum independent set*, SIAM J. Comput., 6 (1977), pp. 537–546.

[16] J. A. Telle and Y. Villanger, *Connecting terminals and 2-disjoint connected subgraphs*, in Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, New York, 2013, pp. 418–428.

[17] P. van't Hof, D. Paulusma, and G. J. Woeginger, *Partitioning graphs into connected parts*, Theoret. Comput. Sci., 410 (2009), pp. 4834–4843.

[18] T. Watanabe, T. Ae, and A. Nakamura, *On the removal of forbidden graphs by edge-deletion or by edge-contraction*, Discrete Appl. Math., 3 (1981), pp. 151–153.

[19] T. Watanabe, T. Ae, and A. Nakamura, *On the NP-hardness of edge-deletion and contraction problems*, Discrete Appl. Math., 6 (1983), pp. 63–78.