Path Contraction Faster than 2^n

Akanksha Agrawal
Hungarian Academy of Sciences, Budapest, Hungary
akanksha@sztaki.hu

Fedor V. Fomin
University of Bergen, Bergen, Norway
fomin@ii.uib.no

Daniel Lokshtanov
University of California Santa Barbara, Santa Barbara, California
daniello@ ucsb.edu

Saket Saurabh
Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Bergen, Norway
UMI ReLax
saket@imsc.res.in

Prafullkumar Tale
Institute of Mathematical Sciences, HBNI, Chennai, India
pptale@imsc.res.in

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 sub-routine in our algorithm for Path Contraction.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Path Contraction, Exact Exponential Time Algorithms, Graph Algorithms, Enumerating Connected Sets, 3-Disjoint Connected Subgraphs

Funding *Saket Saurabh*: This work is supported by the European Research Council (ERC) via grant LOPPRE, reference no. 819416.

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, Odd Cycle Transversal, 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 isomorphic to H. For several families of graphs $\mathcal{F}$, early papers by Watanabe et al. [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: can we 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 $\mathcal{O}^\star((2-\epsilon)^n)$-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, one of the natural question is for which NP-hard problems can we avoid the "brute force search", and say 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.

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), can be "roughly" interpreted as solving P_4-Contraction. (We can use the algorithm for 2-DCS to solve P_4-Contraction.) 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 a $\mathcal{O}^\star(1.933^n)$ 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

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

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 our knowledge, this is the first non-trivial 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 instance, 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 subroutine still serve 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 $\mathcal{W} = \{W_1, W_2, \cdots, W_t\}$ 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 whole graph. The second subroutine searches for a pair of sets in the P_t-witness structure which are very dense. Then it exploit the sparseness of the remaining graph 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 odd/even sets from $\mathcal{W}$ can be bounded. Finally, the fourth subroutine work 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 a generalization 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 is 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 $\mathcal{O}^\star(1.88^n)$.

Due to space limitation, most proofs have been relegated to the attached full version of the paper.

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 notations that will be used throughout. We note that all graphs considered in this article are connected graphs on at least two vertices (unless stated otherwise).

We denote the set of natural numbers by $\mathbb{N}$ (including 0). For $k \in \mathbb{N}$, $[k]$ denotes the set $\{1, 2, \ldots, k\}$. 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 set is given by the ordering of vertices in the path. That is, for a $P_t = (h_1, h_2, \cdots, h_t)$-witness structure, $\mathcal{W} = \{W(h_1), W(h_2), \cdots, W(h_t)\}$ of a graph G, we use the ordered witness structure notation, $(W(h_1), W(h_2), \cdots, W(h_t))$, or simply, $(W_1, W_2, \cdots, W_t)$. We note that we use both unordered and ordered notation, as per the convenience.

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

▷ Observation 2.1. 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 set.

▷ Observation 2.2. For a set U with n elements and a constant $\delta < 1/2$, the number of subsets of U of size at most δn is bounded by $\mathcal{O}^\star([g(\delta)]^n)$, where $g(\delta) = \frac{1}{\delta^\delta \cdot (1-\delta)^{(1-\delta)}}$. Moreover, all such subsets can be enumerated in the same time.

For a graph G, a non-empty 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 give slightly better bounds, but for simplicity, we only use the bounds stated in the following lemmas.)

▶ **Lemma 1.** *For a graph G, a non-empty 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.** *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 later sections.

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

In the 3-DCS problem, the input is 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 non-empty. If either of this set is empty, we can guess a vertex for each of the non-empty sets. Since there are at most n^2 such guesses, it will not affect the running time of our algorithm. Thus, here after we assume that both Z_1 and Z_2 are non-empty sets.

In the following theorem, we state our result regarding 3-DCS.

▶ **Theorem 4.** 3-DCS *admits an algorithm running in time* $\mathcal{O}^\star(1.88^n)$*, where* n *is number of vertices in the input graph.*

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 are 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 sub-routine for enumerating special types of partial solution, 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 $(W_1, W_2, \cdots, W_t)$ 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, $\mathtt{OS} = \cup_{x=1}^{\lceil t/2 \rceil} W_{2x-1}$ 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 1 for an illustration of it.)

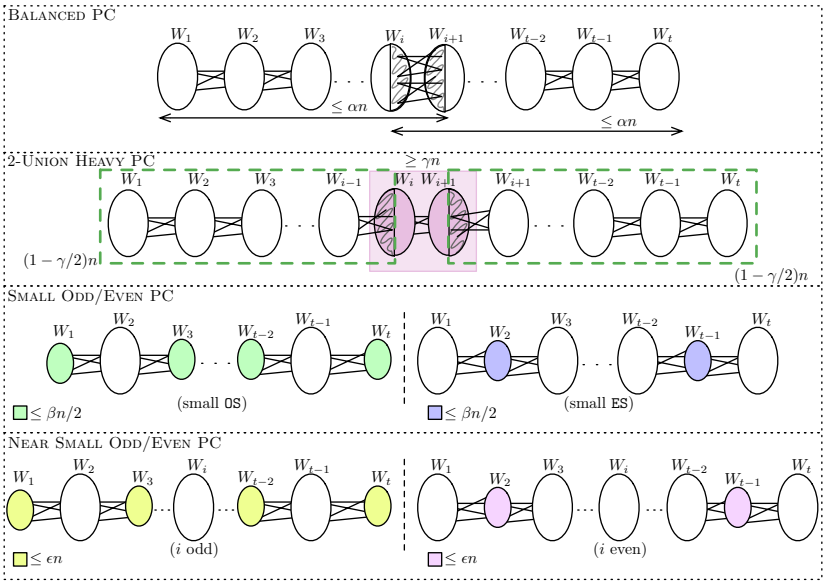

Figure 1 Various subroutines for the algorithm and their usage.

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 have 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 $N[\cup_{j\in[i]}W_j]$ and $N[\cup_{j\in[t]\setminus[i]}W_j]$ 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 "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 term "large/small" will be done by a 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 $|N[\cup_{j\in[i-1]}W_j]|, |N[\cup_{j\in[t]\setminus[i+1]}W_j]| \leq (1-\gamma/2)n$. Moreover, the algorithm will return 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 2.2).

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 $|\texttt{OS}|$ or $|\texttt{ES}|$ 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 Section 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 "nice solution" for all "ρ-small" subset 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 an 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 the end bag. That is, there is a P_{t_S}-witness structure $(W_1, W_2, \cdots W_{t_S})$ 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 $\mathcal{W} = (W_1, W_2, \cdots, W_t)$, 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 ρ-small set. Consider an 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 $\mathcal{O}^\star(2^{\rho n} g(\rho)^n)$. By doing a simple 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 tables entries for our dynamic programming routine, which is used for computation of nice solutions. Let $\mathcal{S}$ be the set of all connected sets S in G, such that $|N[S]| \leq \rho n$. That is, $\mathcal{S} = \{S \subseteq V(G) \mid G[S] \text{ is connected and } |N[S]| \leq \rho n\}$. For each $S \in \mathcal{S}$, we have an entry denoted by $\Gamma[S]$. $\Gamma[S]$ is the largest integer $q \geq 1$ for which $G[S]$ can be contracted to P_q with a P_q-witness structure $\mathcal{W} = (W_1, W_2, \cdots, W_q)$ 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 notations 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

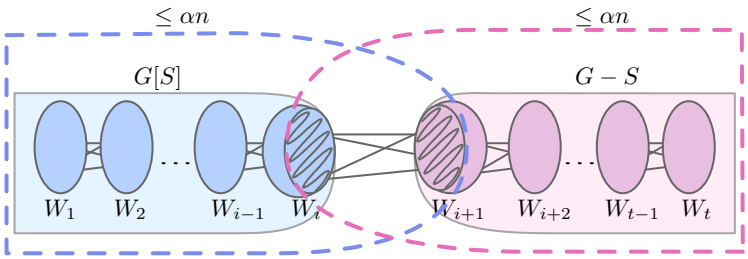

Figure 2 An illustration of construction of the solution using solutions for instances of smaller sizes.

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 integer 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 1. 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 5.** *For each $S \in \mathcal{S}$, the algorithm computes $\Gamma[S]$ correctly.*

▶ **Lemma 6.** *The algorithm presented for* Enum-Partial-PC *runs in time $\mathcal{O}^\star(2^{\rho n})$.*

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 $0 < \alpha \leq 1$.
Output: Largest integer $t \geq 2$ for which G has a P_t-witness structure $\mathcal{W} = (W_1, W_2, \cdots, W_t)$, such that there is $i \in [t]$ with $N[\cup_{j \in [i]} W_j] \leq \alpha n$ and $N[\cup_{j \in [t] \setminus [i]} W_j] \leq \alpha n$. 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 a 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 $(W_1, W_2, \cdots W_{t_S})$ 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 2). Recall that we are looking for the largest integer t, such that G is contractible to P_t, with $\mathcal{W} = (W_1, W_2, \cdots, W_t)$ as a P_t-witness structure of G, such that there is $i \in [t]$ with

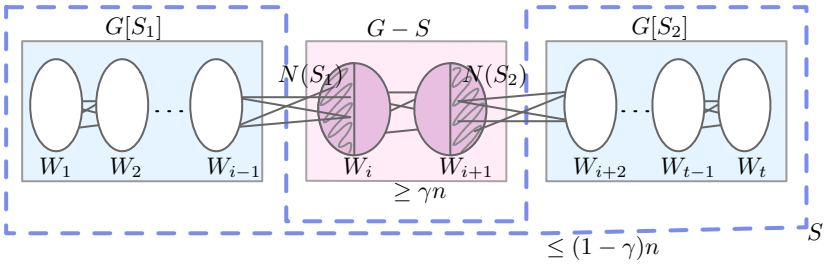

Figure 3 An intuitive illustration of the algorithm for 2-Union Heavy PC.

$|\cup_{j\in[i+1]} W_j| \leq \alpha n$ and $|\cup_{j\in[t]\setminus[i-1]} W_j| \leq \alpha n$. Let $S = \cup_{j\in[i]} W_j$. As $|\cup_{j\in[i+1]} W_j| \leq \alpha n$ 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 algorithm from Section 4.1. Let $\mathcal{S}$ be the set of all connected sets S in G, such that $|N[S]| \leq \alpha n$. That is, $\mathcal{S} = \{S \subseteq V(G) \mid G[S] \text{ is connected and } |N[S]| \leq \alpha n\}$. 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 7.** *The algorithm presented for* Balanced PC *is correct.*

▶ **Lemma 8.** *The algorithm presented for* Balanced PC *runs in time* $\mathcal{O}^\star(2^{\alpha n})$.

4.3 Algorithm for 2-Union Heavy PC

We formally define the problem 2-Union Heavy PC in the following (also see Figure 1).

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 $\mathcal{W} = (W_1, W_2, \cdots, W_t)$, 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) $|N[\cup_{j\in[i-1]} W_j]|, |N[\cup_{j\in[t]\setminus[i+1]} W_j]| \leq (1 - \gamma/2)n$. Moreover, if no such t exists, then output 2.

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].

Let (G, γ) be an instance of 2-Union Heavy PC. We start by explaining the intuitive idea behind our algorithm (see Figure 3). Consider a P_t-witness structure $\mathcal{W} = (W_1, W_2, \cdots, W_t)$ of G, 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) $|N[\cup_{j\in[i-1]} W_j]|, |N[\cup_{j\in[t]\setminus[i+1]} W_j]| \leq (1 - \gamma/2)n$. Let $S = W_i \cup W_{i+1}$. As $W_i \cup W_{i+1}$ is "large", the number of vertices in $S = V(G) \setminus (W_i \cup W_{i+1})$ is "small". That is, we can bound $|S|$ by $(1-\gamma)n$. Note that $G[S]$ has exactly two connected components, which we

denote by $G[S_1]$ and $G[S_2]$. Also note that $N[S_1], N[S_2] \leq (1-\gamma/2)n$. The algorithm starts by enumerating all such "potential candidates" for S. As for each of the two components of $G[S]$, the sizes of $N[S_1]$ and $N[S_2]$ can be bounded, the algorithm computes the "optimum solution" for them using the algorithm for Enum-Partial-PC. In the above we use the algorithm for Enum-Partial-PC because we are only interested in those solutions where the vertices of $\Phi(S_1)$ and $\Phi(S_2)$ are contained in one of the "end bags" of their respective solutions. Now we see how we can use these solutions to obtain the solution for the whole graph. Note that we have to "split" vertices in $V(G) \setminus S$ into two "connected sets", where the first set must contain all the vertices from $N(S_1)$ and the second set must contain all the vertices from $N(S_1)$. For the above we employ the algorithm for 2-Disjoint Connected Subgraphs (see Section 3 for its definition) by 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 $G[S_1], G[S_2]$, s.t. $|N[S_1]|, |N[S_2]| \leq (1-\gamma/2)n\}$. 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 computes 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 $\mathcal{W} = (W_1, W_2, \cdots, W_q)$ 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 $|N[S_1]|, N[S_1]| \leq (1-\gamma/2)n$. 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 $\mathcal{S}$. Finally, the algorithm outputs t. This completes the description of the algorithm.

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

▶ **Lemma 9.** *The algorithm presented for* 2-Union Heavy PC *is correct.*

▶ **Lemma 10.** *The algorithm presented for* 2-Union Heavy PC *runs in time* $\mathcal{O}^\star(2^{(1-\gamma/2)n} + c^n)$*, where* $c = \max_{\gamma \leq \delta \leq 1}\{1.7804^\delta \cdot g(1-\delta)\}$.

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 $\mathcal{W} = (W_1, W_2, \cdots, W_t)$ as a P_t-witness structure of G, such that $|\texttt{OS}_\mathcal{W}| \leq \beta n/2$ or $|\texttt{ES}_\mathcal{W}| \leq \beta n/2$, where $\texttt{OS}_\mathcal{W} = \cup_{i \in [\lceil t/2 \rceil]} W_{2i-1}$ and $\texttt{ES}_\mathcal{W} = \cup_{i \in [\lfloor t/2 \rfloor]} W_{2i}$.

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 $\texttt{OS}_\mathcal{W}$ (resp. $\texttt{ES}_\mathcal{W}$), 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.

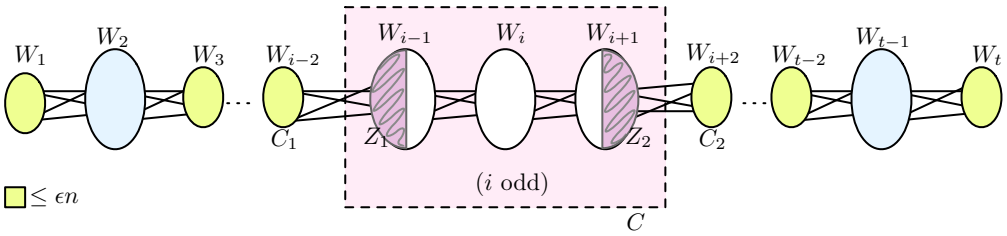

Figure 4 An intuitive illustration of the algorithm for Near Small Odd/Even PC.

We now move to 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 2.2. 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 11.** *The algorithm presented for* Small Odd/Even PC *is correct and runs in time* $\mathcal{O}^\star(g(\beta/2)^n)$.

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 1).

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 $\mathcal{W} = (W_1, W_2, \cdots, W_t)$ of G, for which there is $i \in \{2, 3, \cdots, t-1\}$, 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, $\mathtt{OS}_{\mathcal{W}} = \cup_{i \in [\lceil t/2 \rceil]} W_{2i-1}$ and $\mathtt{ES}_{\mathcal{W}} = \cup_{i \in [\lfloor t/2 \rfloor]} W_{2i}$. 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 multiplicative factor will be due enumeration of sets, and the first term will be due to calls made to the algorithm for 3-Disjoint Connected Subgraphs, from Section 3.

Let (G, ϵ) be an instance of Near Small Odd/Even PC. We start by explaining the intuitive idea behind our algorithm (see Figure 4). Consider a P_t-witness structure $\mathcal{W} = (W_1, W_2, \cdots, W_t)$ of G, for which there is $i \in \{2, 3, \cdots, t-2\}$, 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, $\mathtt{OS}_{\mathcal{W}} = \cup_{i \in [\lceil t/2 \rceil]} W_{2i-1}$ and $\mathtt{ES}_{\mathcal{W}} = \cup_{i \in [\lfloor t/2 \rfloor]} W_{2i}$. 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 4 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, the components of $G[S]$ must each 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. 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 12.** *The algorithm presented for* Near Small Odd/Even PC *is correct.*

▶ **Lemma 13.** *The algorithm presented for* Near Small Odd/Even PC *runs in time* $\mathcal{O}^\star(c^n)$*, where* $c = \max_{0 \leq \delta \leq \epsilon}\{1.88^{(1-\delta)} \cdot g(\delta)\}$.

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 14.** 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.*

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 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.

References

1. Takao Asano and Tomio Hirata. Edge-Contraction Problems. *Journal of Computer and System Sciences*, 26(2):197–208, 1983.
2. Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014.
3. Andries Evert Brouwer and Hendrik Jan Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987.
4. Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.
5. Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Solving the 2-disjoint connected subgraphs problem faster than 2^n. *Algorithmica*, 70(2):195–207, 2014.
6. Konrad K Dabrowski and Daniël Paulusma. Contracting bipartite graphs to paths and cycles. *Information Processing Letters*, 127:37–42, 2017.
7. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
8. Jiří Fiala, Marcin Kamiński, and Daniël Paulusma. A note on contracting claw-free graphs. *Discrete Mathematics and Theoretical Computer Science*, 15(2):223–232, 2013.
9. Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012.
10. Pinar Heggernes, Pim van 't Hof, Benjamin Lévêque, and Christophe Paul. Contracting chordal graphs and bipartite graphs to paths and trees. *Discrete Applied Mathematics*, 164:444–449, 2014.
11. Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
12. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
13. Walter Kern and Daniel Paulusma. Contracting to a longest path in H-free graphs. *arXiv preprint arXiv:1810.01542*, 2018.
14. Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018.
15. Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977.
16. Jan Arne Telle and Yngve Villanger. Connecting terminals and 2-disjoint connected subgraphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 418–428. Springer, 2013.
17. Pim van't Hof, Daniël Paulusma, and Gerhard J Woeginger. Partitioning graphs into connected parts. *Theoretical Computer Science*, 410(47-49):4834–4843, 2009.
18. Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discrete Applied Mathematics*, 3(2):151–153, 1981.
19. Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the NP-hardness of edge-deletion and contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983.