On the Complexity of Finding a Sparse Connected Spanning Subgraph in a Non-Uniform Failure Model

Matthias Bentert
Department of Informatics, University of Bergen, Norway

Jannik Schestag
Faculteit Elektrotechniek, Wiskunde en Informatica, TU Delft, The Netherlands
Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany

Frank Sommer
Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany

Abstract

We study a generalization of the classic Spanning Tree problem that allows for a non-uniform failure model. More precisely, edges are either *safe* or *unsafe* and we assume that failures only affect unsafe edges. In Unweighted Flexible Graph Connectivity we are given an undirected graph $G = (V, E)$ in which the edge set E is partitioned into a set S of safe edges and a set U of unsafe edges and the task is to find a set T of at most k edges such that $T - \{u\}$ is connected and spans V for any unsafe edge $u \in T$. Unweighted Flexible Graph Connectivity generalizes both Spanning Tree and Hamiltonian Cycle. We study Unweighted Flexible Graph Connectivity in terms of fixed-parameter tractability (FPT). We show an almost complete dichotomy on which parameters lead to fixed-parameter tractability and which lead to hardness. To this end, we obtain FPT-time algorithms with respect to the vertex deletion distance to cluster graphs and with respect to the treewidth. By exploiting the close relationship to Hamiltonian Cycle, we show that FPT-time algorithms for many smaller parameters are unlikely under standard parameterized complexity assumptions. Regarding problem-specific parameters, we observe that Unweighted Flexible Graph Connectivity admits an FPT-time algorithm when parameterized by the number of unsafe edges. Furthermore, we investigate a below-upper-bound parameter for the number of edges of a solution. We show that this parameter also leads to an FPT-time algorithm.

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

Keywords and phrases Flexible graph connectivity, NP-hard problem, parameterized complexity, below-guarantee parameterization, treewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.4

Related Version A continuously updated version of the paper is available at `https://arxiv.org/abs/2308.04575`.

Funding *Matthias Bentert*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
Jannik Schestag: Supported by the German Academic Exchange Service (DAAD), project 57556279.
Frank Sommer: Supported by the Deutsche Forschungsgemeinschaft (DFG), project EAGR, KO 3669/6-1.

Acknowledgements This work was initiated at the research retreat of the Algorithmics and Computational Complexity group of TU Berlin held in Darlingerode in September 2022.

1 Introduction

Computing a spanning tree is a fundamental task in computer science with a huge variety of applications in network design [14] and clustering problems [29]. In Spanning Tree, one is given a graph G, and the aim is to find a set $T \subseteq E(G)$ of minimal size such that each pair of vertices in G is connected via edges in T. It is well known that Spanning Tree can be solved in polynomial time [22, 25]. This classic spanning tree model has, however, a major drawback: all edges are seen as equal. In many scenarios, for example in the construction of supply chains [27], this is not sufficient: some connections might be more fragile than others. To overcome this issue, several different network-design and connectivity problems are studied with additional robustness constraints [13, 26]. In this work, we continue this line of research and investigate a graph model in which the edge set is partitioned into *safe* edges S and *unsafe* edges U [1, 2, 8]. In contrast to several other variants of robust connectivity, these two edge types enable us to model a non-uniform failure scenario [1]. With these types at hand, we can relax the model of a spanning tree in the sense that one unsafe edge may fail. An edge set T of a graph $G = (V, S, U)$ is an *unsafe spanning connected subgraph* if a) T is spanning for V, and b) $T - \{e\}$ is connected for each unsafe edge $e \in T$ [1]. This leads to the following problem:

Unweighted Flexible Graph Connectivity (UFGC)

Input: A graph $G = (V, S, U)$ and an integer k.
Question: Is there an unsafe spanning connected subgraph T of G with $|T| \leq k$?

In the following, we refer to T as a solution. UFGC generalizes several well-studied classic graph problems. Examples include 2-Edge Connected Spanning Subgraph [20, 26] ($S = \emptyset$) and Hamiltonian Cycle [3, 19] ($S = \emptyset$ and $k = |V(G)|$). Thus, in sharp contrast to the classic Spanning tree problem, UFGC is NP-hard.

Related Work. If the graph is additionally equipped with an edge-cost function, the corresponding problem, in which one aims to find a solution of minimum total weight, is known as Flexible Graph Connectivity [1, 8]. Flexible Graph Connectivity is mainly studied in terms of approximation: Adjiashvili et al. [1] provided a polynomial-time 2.527-approximation algorithm which was improved by Boyd et al. [8] to a polynomial-time 2-approximation. Recently, a generalization (p, q)-Flexible Graph Connectivity was introduced [8] and studied in terms of approximation. In this model, up to p unsafe edges may fail and the result shall be q-edge connected. Clearly, Flexible Graph Connectivity is the special case where $p = 1$ and $q = 1$. Boyd et al. [8] provided a $(q + 1)$-approximation for the case $p = 1$ and a $\mathcal{O}(q \log(n))$ approximation for the general case. For the special case of $p = 2$, an improved approximation of $\mathcal{O}(q)$ was provided by Chekuri and Jain [9], and for the special case of $q = 1$ a $\mathcal{O}(1)$ approximation was shown by Bansal et al. [6].

Our model with safe and unsafe edges is based on prior work by e.g. Adjiashvili et al., who studied the approximability of the classic (s, t)-Path and (s, t)-Flow problems in this model [2]. Even previous to Adjiashvili et al. [2], some studies already indirectly investigated problems in this setting: There have been some studies of classic graph-connectivity problems where one wishes for the stronger requirement of 2-edge connectivity. In terms of our setting, this corresponds to the case that all edges are unsafe. One prominent example is the aforementioned 2-Edge Connected Spanning Subgraph [10, 21, 26]. There exists a 4/3-approximation [10, 21, 26], and an 6/5-approximation in cubic graphs [10]. Furthermore, for its generalization q-Edge Connected Spanning Subgraph approximation algorithms

of ratio $1+1/(2q)+\mathcal{O}(1/(q^2))$ are known [17]. q-Edge Connected Spanning Subgraph is also studied in terms of parameterized complexity: Basavaraju et al. [4] provided an FPT-time algorithm with respect to the number of edges deleted from G to obtain a solution. A variant of 2-Edge Connected Spanning Subgraph in digraphs was studied by Bang-Jensen and Yeo [5]. They provided an FPT-time algorithm for a solution-size related parameter below an upper bound.

Our Results. We study the parameterized complexity of UFGC with respect to many natural parameters. We investigate problem specific and also structural graph parameters. A first idea to solve UFGC might be the following: merge each safe component into a single vertex by computing an arbitrary spanning tree of this component and then use existing algorithms for 2-Edge Connected Spanning Subgraph to solve the remaining instance. This approach, however, does not yield a minimal solution: Consider a C_4 on unsafe edges where we add one safe edge. Then, the unique minimal solution consists of all four unsafe edges. Hence, we need new techniques to solve UFGC optimally.

In terms of problem specific parameters, we first study parameterization by the number $|U|$ of unsafe edges. We provide an FPT-time algorithm for $|U|$ (Proposition 3.1), by exploiting the fact that Disjoint Subgraphs admits an FPT-time algorithm. Second, we investigate parameterizations above a lower bound and below an upper bound for the solution size. For the former parameterization, we obtain hardness due to the connection to Hamiltonian Cycle and for the latter we obtain an FPT-time algorithm (Theorem 3.2). Our algorithm is inspired by an algorithm of Bang-Jensen and Yeo [5] for Minimum Spanning Strong Subgraph. In this problem one is given a digraph D and one wants to find a minimum spanning strong digraph of D. The main technical hurdle in our adaption is that in UFGC we have two different edge types which have to be treated differently compared with the problem studied by Bang-Jensen and Yeo [5].

We also study parameterization by structural graph parameters; see Figure 1 towards a dichotomy for UFGC. For many parameters such as maximum degree and domination number we obtain para-NP-hardness, due to the connection to Hamiltonian Cycle. For other parameters, we obtain FPT-time algorithms. For example, for the treewidth tw of the input graph and the (vertex) deletion distance to cluster graphs. With these two algorithms at hand, we obtain an almost complete border between parameters which allow for FPT-time algorithms and those that do not.

For the treewidth tw, we present an FPT-time algorithm based on dynamic programming with running time $n \cdot 2^{\mathcal{O}(\mathrm{tw}\log(\mathrm{tw}))}$ (Proposition 4.1). In order to achieve this running time, we define and exploit an encoding of size $\mathrm{tw}^{\mathrm{tw}}$ to check all possibilities on how the current bag of the tree decomposition is connected with the already considered vertices. Finally, we show our main technical result: UFGC parameterized by the vertex-deletion distance to cluster graphs admits an FPT-time algorithm (Theorem 4.2). Therein, we use a combination of the following two main ingredients: First, given a modulator K, that is, a set of vertices such that $G' = G - K$ is a cluster graph, we can safely connect all vertices in K using vertices from $O(|K|)$ cliques in G' in any solution. We call these cliques the *backbone* of the solution and we can guess in FPT time the structure of the backbone. Second, we show how to compute the size of a smallest solution that implements a backbone in FPT time using algorithms for both Maximum Bipartite Matching and Disjoint Subgraphs in the process. Due to lack of space, several proofs (marked with ($\star$)) are deferred to the full version.

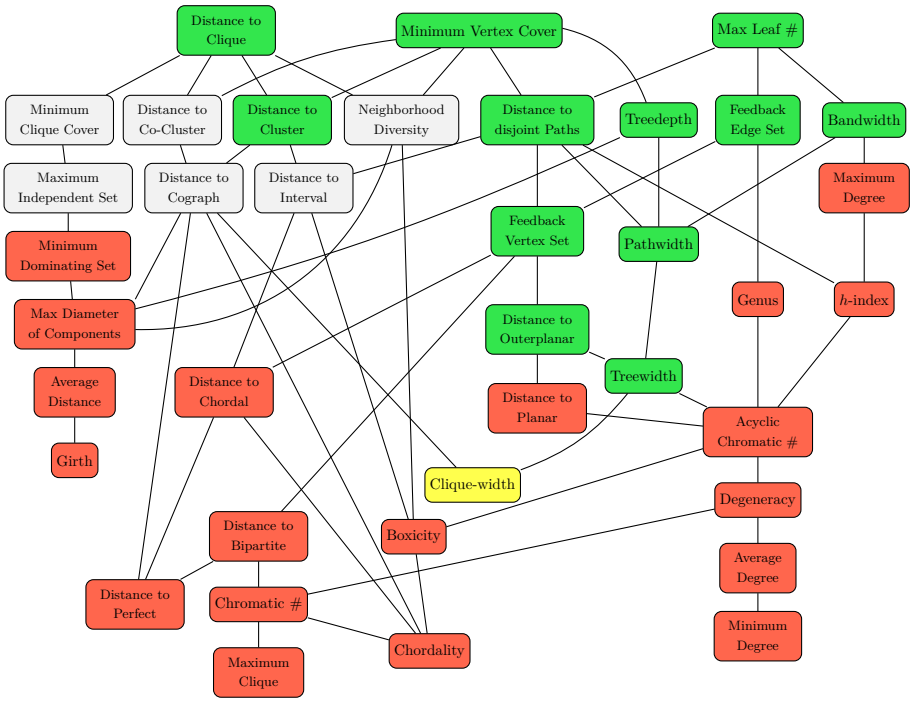

Figure 1 The relations between structural graph parameters and our respective results for UFGC. A parameter k is marked green (●) if UFGC admits an FPT-time algorithm for k, yellow (●) if it is W[1]-hard with respect to k, and red (●) if it is NP-hard for constant k (para-NP-hard). We do not know the status for parameters with white boxes. An edge from a parameter α to a parameter β below α means that there is a function f such that $\beta \leq f(\alpha)$ in every graph. Hardness results for α imply the same hardness results for β and an FPT-time algorithm for β implies an FPT-time algorithm for α.

Preliminaries. For $n \in \mathbb{N}$, by $[n]$ we denote the set $\{1, \ldots, n\}$. Throughout this work, all logarithms have 2 as their base. For a graph $G = (V, S, U)$, we denote by $V(G)$ and $E(G) \coloneqq S \cup U$ its *vertex set* and *edge set*, respectively. Furthermore, by $n \coloneqq |V(G)|$ we denote the number of vertices. Let $Z \subseteq V(G)$ be a vertex set. By $G[Z]$ we denote the *subgraph induced* by Z, and by $G - Z \coloneqq G[V(G) \setminus Z]$ we denote the graph obtained by *removing* the vertices of Z. We denote by $N_G(Z) \coloneqq \{y \in V(G) \setminus Z : \{y, z\} \in E(G), z \in Z\}$ and $N_G[Z] \coloneqq N_G(Z) \cup Z$, the *open* and *closed neighborhood* of Z, respectively. For all these notations, whenever Z is a singleton $\{z\}$ we may write z instead of $\{z\}$. We may drop the subscript $\cdot_G$ when it is clear from the context. Let $u, v \in V(G)$. We say that u and v are *safely connected* if there exists a path from u to v using only safe edges or if there exist two paths P_1 and P_2 from u to v such that $E(P_1) \cap E(P_2) \subseteq S$. We say that u and v are *unsafe connected* if u and v are not safely connected and if there exists a path from u to v. For more details on graph notation we refer to the standard monograph [28].

A parameterized problem is *fixed-parameter tractable* if every instance (I, k) can be solved in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time for some computable function f. For more details on parameterized complexity, we refer to the standard monographs [11, 12].

2 Basic Observations

In this section, we explore the aforementioned connection between UFGC and Hamiltonian Cycle and some implications thereof. To this end, note that if the graph contains no safe edges, then each vertex needs to be contained in at least one cycle in the solution. We show that a solution of size n to UFGC (if such a solution exists) corresponds to a Hamiltonian cycle.

▶ **Observation 2.1** (⋆). *Let G be a graph and let $I := (G' = (V(G), \emptyset, E(G)), n)$. Then G contains a Hamiltonian cycle if and only if I is a yes-instance of UFGC.*

Since Observation 2.1 describes a polynomial-time reduction from Hamiltonian Cycle to UFGC, we can directly transfer any NP-hardness results for Hamiltonian Cycle on restricted graph classes to UFGC for the special case that there are no safe edges. It is known that Hamiltonian Cycle remains NP-hard on subcubic bipartite planar graphs [3], on split graphs [19], and on graphs with constant maximum degree [18]. Moreover, Hamiltonian Cycle is NP-hard for graphs with exactly one universal vertex, as shown next.

▶ **Observation 2.2** (⋆). *Hamiltonian Cycle is NP-hard even if there is exactly one universal vertex.*

We obtain the following simple corollary for UFGC.

▶ **Corollary 2.3.** *UFGC is NP-hard, even if there are no safe edges, k equals n, and the graph G a) is subcubic, planar and bipartite, b) is a split graph, c) has domination number one, or d) has constant maximum degree.*

Moreover, since Hamiltonian Cycle is W[1]-hard parameterized by clique-width cw [15], and cannot be solved in $f(\mathrm{cw}) \cdot n^{o(\mathrm{cw})}$ time [16], we obtain the following.

▶ **Corollary 2.4.** *UFGC is W[1]-hard with respect to the clique-width cw, even if $S = \emptyset$ and $k = n$. Moreover, this restricted version cannot be solved in $f(\mathrm{cw}) \cdot n^{o(\mathrm{cw})}$ time.*

▶ **Observation 2.5** (⋆). *UFGC is NP-hard even if the bisection width is one.*

3 Problem-Specific Parameters

In this section, we study problem-specific parameters. In particular, we show that UFGC is fixed-parameter tractable when parameterized by the number of unsafe edges in the input graph and we study some *above-lower-bound* and *below-upper-bound* parameterizations for the solution size. Such parameters are frequently studied [23, 24]. However, since Observation 2.1 shows hardness for UFGC where $k = n$ and since each solution has at least $n - 1$ edges, the parameterization above the lower bound of $n - 1$ is hopeless. Hence, we focus on a parameterization below an upper bound. More precisely, we show that an optimal solution of G consists of at most $2n - 4$ edges. Then, we aim to find a solution for G with $k = 2n - 4 - \ell$ edges for small values of ℓ.

▶ **Proposition 3.1** (⋆). *UFGC is fixed-parameter tractable when parameterized by $|U|$.*

We continue by presenting an FPT-time algorithm for the below lower bound parameter $\ell := 2n - 4 - k$ for UFGC.

▶ **Theorem 3.2** (⋆). *UFGC can be solved in $\ell^{8\ell} \cdot \mathrm{poly}(n)$ time, where $\ell := 2n - 4 - k$.*

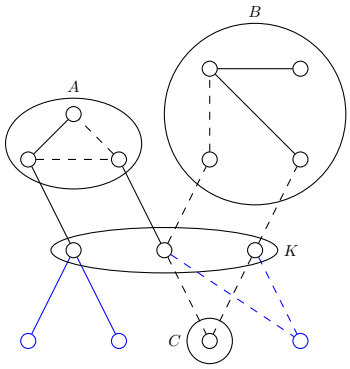

Figure 2 An example of a solution. Safe edges are depicted with solid lines and unsafe edges are depicted with dashed lines. The modulator K and three connecting components A, B, and C of a minimal backbone are drawn in black. All other vertices (drawn in blue) are not part of the minimal backbone as all vertices in K are already safely connected in the black subgraph. The connecting component A is a cyclic component and B and C are usual connecting components.

4 Structural Graph Parameters

In this section, we study two structural graph parameters. We start by giving an FPT-time algorithm for the parameter treewidth tw. Afterwards, we develop a far more intricate FPT-time algorithm for the parameter (vertex-deletion) distance to cluster graphs. Recall that a cluster graph is a graph in which each connected component is a clique.

▶ **Proposition 4.1** (⋆). *UFGC parameterized by* tw *is solvable in* $\mathcal{O}(n \cdot 2^{37\,\mathrm{tw}\log(\mathrm{tw})})$ *time.*

We continue with the algorithm for distance to cluster graphs.

▶ **Theorem 4.2.** *UFGC parameterized by the (vertex-deletion) distance* ℓ *to cluster graphs can be solved in* $f(\ell) \cdot n^3$ *time for some computable function* f*.*

Proof. We start by computing a set K of ℓ vertices in $\mathcal{O}(3^\ell \cdot n^3)$ time such that $G' = G[V \setminus K]$ is a cluster graph as follows. Note that a graph G is a cluster graph if and only if it does not contain an induced P_3, that is, three vertices a, b, and c with $\{a, b\}, \{b, c\} \in E(G)$ and $\{a, c\} \notin E(G)$. Hence, we can find an induced P_3 in $\mathcal{O}(n^3)$ if it exists and then branch on which of the three vertices in the P_3 to include in K. Note that K needs to contain at least one of the three vertices and the resulting search tree has therefore depth ℓ and size 3^ℓ.

We next give a few definitions required to give a more detailed description of the algorithm afterwards. We call a subgraph of a solution $H = (V_H, S_H, U_H)$ a *backbone* if it contains all vertices in K and each pair of vertices in V_H is safely connected in H. See Figure 2 for an example. Given such a backbone $H = (V_H, S_H, U_H)$, a *connecting component* is a connected component in $H' = H[V_H \setminus K]$. We say that a backbone is minimal if the removal of any connecting component results in some pair of vertices in K being not safely connected anymore. We distinguish between two types of connecting components: *cyclic* and *usual*. A cyclic connecting component is a cycle (with some connections to vertices in K). Note that the number of edges in a cycle equals its number of vertices. A usual connecting component is a tree. Its number of edges is one less than its number of vertices but if it contains unsafe edges, then not all vertices are safely connected within the connecting component. Observe that we can indeed assume that each connecting component is either usual or cyclic as any connecting component that is not a tree contains at least as many edges as vertices. Hence,

we can replace such a connecting component by any cycle through all of its vertices. Such a cycle exists within any clique of size at least three (any permutation of the vertices results in a cycle and any connecting component inside a clique of size at most two is a tree) and any pair of vertices in a cyclic connecting component is safely connected.

Next, we distinguish between three types of cliques in G'. To this end, let C be a connected component in G', that is, a clique in the cluster graph. The three types are based on the edges between C and K (note that all edges between vertices in C and the rest of the graph are to vertices in K) and on the connected components within C if we ignore all unsafe edges. We call such components *strong components*. A clique C is *strong*, if each strong component in C contains at least one vertex with a safe edge to a vertex in K. In this case, we can add C to the backbone using $|C|$ (safe) edges (a maximal spanning forest plus for each strong component one edge connecting it to K). The second type, we call *weak cliques*. A weak clique is not a strong clique but it is connected to K by a safe edge or by two unsafe edges with different endpoints in C. Since weak cliques are not strong cliques, there is some strong component in them, which is not connected to K via only safe edges. Hence, to safely connect this strong component L to K, we require at least $|L| + 1$ edges. Since we require at least $|C \setminus L|$ edges to connect the remaining vertices, we require at least $|C| + 1$ edges to safely connect C to the backbone. For weak cliques, $|C| + 1$ edges are sufficient as we can use a) a safe edge between C and K and a Hamiltonian cycle in C or b) two unsafe edges to different vertices in C and any Hamiltonian path between the two endpoints within C. We call the third type of clique *singletons*. A singleton is only connected to K by unsafe edges and all of these edges have the same endpoint in C. Note that if there is at most one unsafe edge between K and C (and assuming that $K \neq \emptyset$), then there cannot be a solution as C cannot be safely connected to K. We call them singletons because we can reduce such a clique to a single vertex in a preprocessing step. Since all connections to K are through one vertex $v \in C$, we have to safely connect all vertices in C to v. This can either be done via a spanning tree consisting only of safe edges (if such a tree exists) or via any Hamiltonian cycle otherwise[1]. Which case applies can be checked in linear time by checking whether there is exactly one strong component in C.

We are now in a position to describe the algorithm. First, we guess which edges between vertices in K belong to a solution T and the number p of connecting components in a minimal backbone H of T. Note that $p \leq 2\ell$ since any connecting component in H provides an (unsafe) connection between two vertices in K and any cycle of unsafe connections implies also safe connections. Hence, in the worst case each cycle is of length two and we require $2\ell - 2$ connections to implement a "safe spanning tree" between the vertices in K. Next, we guess the structure of H, that is, for each connecting component P in H, we guess the following (see also Figure 3).

1. Which vertices in K are adjacent to vertices in P in T?
2. How large is the set Y of vertices in P that are neighbors to vertices in K in T?[2]
3. Which edges between K and Y are contained in T?[3]
4. Which pairs of vertices in Y are safely connected within Y? (That is, a partition of the vertices in Y)
5. Is P a usual or a cyclic component?

[1] The Hamiltonian cycle exists if there are at least three vertices in C. If $|C| = 2$ and there is only an unsafe edge between the two vertices, then there cannot be a solution and we can return *no*.

[2] Note that $|Y| \leq 2\ell$ by the same argument that shows $p \leq 2\ell$.

[3] Note that we something like "there are three vertices $y_1, y_2, y_3 \in Y$ and the solution contains the safe edge $\{u, y_1\}$ and the unsafe edges $\{v, y_2\}, \{v, y_3\}$, and $\{w, y_3\}$". However, we do not guess which vertex in the input graph is a vertex in Y. For an example, we refer to Figure 3.

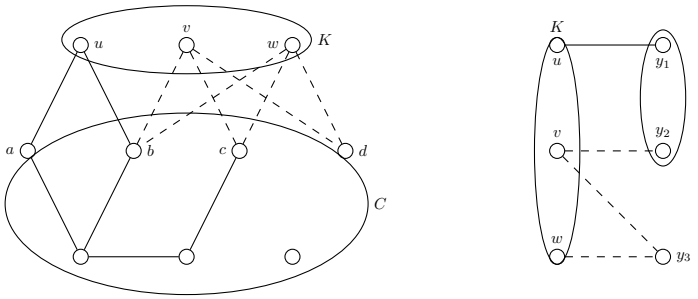

Figure 3 The left side depicts the modulator K and one clique C in $G - K$. Safe edges are depicted with solid lines and unsafe edges are depicted with dashed lines. To reduce visual clutter, we do not show the unsafe edges between two vertices in C. The right side depicts one possible guess for a connecting component. It contains three vertices y_1, y_2, and y_3, some edges between K and a partition of the guessed vertices (in our case y_1 and y_2 are guessed to be safely connected within C). Note that in the graph on the left side there are many different possibilities to realize the guess on the right side. One possibility is $y_1 = a$, $y_2 = b$ and $y_3 = d$. A second possibility is $y_1 = b$, $y_2 = c$ and $y_3 = d$ and a third possibility $y_1 = a$, $y_2 = c$ and $y_3 = b$.

Moreover, we guess which connecting components of the minimal backbone are contained in the same clique in G', that is, we guess a partition of the p connecting components. Finally, we guess for each part of the partition the type of clique that contains the connecting components and how the rest of the clique (that is, all vertices in the clique that are not contained in any connecting component) is connected to the minimal backbone.

We distinguish between the following three types of connections. To this end, let C be a clique in G' that hosts at least one connecting component of H and let $C' = C \setminus V_H$ be the set of vertices that are not contained in a connecting component. If there is a connecting component in $\mathcal{P}$ that contains at least one unsafe edge, then we can replace this edge with a Hamiltonian path through all vertices in C'. If this is the case or if $C' = \emptyset$, then we say that C' is *empty*. Otherwise, if we can safely connect all vertices in C' to the backbone using $|C'|$ edges, then we say that C' is *efficiently connected* to the backbone. This is the case if each vertex in C' is contained in a strong component in C that contains a) a vertex in some $P_i \in \mathcal{P}$ or b) a vertex with an incident safe edge to a vertex in K. If neither of the two cases above applies, then we require at least $|C'| + 1$ edges to connect the vertices in C' to the backbone. Note that in this case, we can always find a path through all vertices in C' and connect the two ends to any vertex in $C \setminus C'$. We call this type of connection *inefficient*.

Observe that if two solutions lead to exactly the same set of guesses, then they have the same number of edges[4] as the difference between the number of edges and vertices in their minimal backbones and the types of the remaining cliques (both of cliques containing parts of the minimal backbone and those which do not) is the same for both solutions. As argued above, the difference between the number of edges and vertices in these cliques in a solution is completely determined by their type.

It remains to show how to check whether a guess leads to a solution and to analyze the running time. Towards the former, we first show how to test whether a given clique C of a guessed type can host a guessed set $\mathcal{P} = \{P_1, \ldots, P_c\}$ of connecting components such that the rest of the clique has the guessed connection type. We can handle most combinations of clique type and connection type with a general approach. One special case has to be treated differently: The connection type is efficient, each connecting component $P_i \in \mathcal{P}$ is

[4] Assuming that the solutions are minimal, that is, they do not contain edges whose removal yields a smaller solution.

a usual component which safely connects all respective vertices in Y and the clique C is a weak clique. First, we describe why this case is different from the others. Then, we show how to handle this special case and how to handle all other cases. The difference is that if the connection type is "C' is empty" or inefficient, then we can pretty much ignore the connection type as we can always greedily find a solution independent of the connecting component. If one of the connecting components in $\mathcal{P}$ is a cyclic component or a usual component with at least one unsafe edge in it, then the connection type is "C' is empty".[5] The same is true if the clique C is a singleton. If C is a strong clique, then we can also ignore C' as we can always greedily find an efficient connection independent of the connecting component. Only if the connection type is efficient, $\mathcal{P}$ only consists of usual components which safely connect all respective vertices in Y, and C is a weak clique, then we somehow need to "hit all strong components in C" which do not have a safe edge to a vertex in K using the connecting components.

The next step in this proof is to describe our algorithm to check if a clique C can host a set $\mathcal{P} = \{P_1, P_2, \dots, P_c\}$ of connecting components where each P_i is a usual connecting component consisting only of safe edges. Due to space constraints, we only show this algorithm in this extended abstract; the running time analysis is deferred to the full version. Informally speaking, we first show that each P_i is contained in a different strong component in C. We then use Maximum Bipartite Matching to first check in which strong components V_j in C each P_i can be contained in. We then check whether we can assign each P_i to some V_j such that P_i can be contained in V_j and each strong component V_j which does not have a safe edge to a vertex in K contains some P_i.

▷ Claim 4.3 (⋆). We can check in $\mathcal{O}(\ell^2 \cdot n^3)$ time whether a clique C can host a set $\mathcal{P} = \{P_1, P_2, \dots, P_c\}$ of connecting components where each P_i is a usual connecting component consisting only of safe edges.

Next, we present the general algorithm for all remaining cases, that is, the connection type is not efficient, the clique C is not a weak clique, or the set of connecting components contains an unsafe edge. Here, we can ignore the connection type and we only need to check whether the guessed set $\mathcal{P}$ of connecting components can be hosted in a clique C. Since C is a clique, any pair of vertices in C is connected by an edge. Hence, we can trivially connect any set of t vertices with unsafe connections using $t-1$ edges. We only need to check two points:

1. Is there for each vertex in Y (the neighbors of vertices in K in C) a distinct vertex in C with all the required edges to vertices in K and
2. can all sets of vertices that are guessed to be pairwise connected via paths of safe edges inside C be connected in this way?

Note that the latter point is unfortunately not as simple as checking whether all vertices belong to the same strong component in C as the example in Figure 4 shows. We solve both points as follows. We built a graph with two vertices y, y' for each vertex $y \in Y$ and a vertex v for each vertex $v \in C$. There are edges $\{y, v\}$ and $\{y', v\}$ if the following holds. For each edge $\{x, y\}$ with $x \in K$ that is guessed to be in the solution, the edge $\{x, v\}$ is contained in the input graph G and the edge $\{x, v\}$ is safe if and only if the edge $\{x, y\}$ is guessed to be safe. Moreover, there is an edge between two vertices $u, v \in C$ if and only

[5] We may assume that there is only one cyclic component in $\mathcal{P}$ as we can always merge two cyclic components in one clique into one cyclic component. We need to consider the special case where the cyclic component contains exactly two vertices with edges to vertices in K as we need to ensure that there is at least one additional vertex not contained in any of the other connecting components in $\mathcal{P}$.

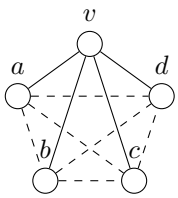

Figure 4 A clique with five vertices is shown. Safe edges are depicted with solid lines and unsafe edges are depicted with dashed lines. Suppose we want to connect a to b and c to d using only safe edges. All vertices belong to the same strong component in C but the only solution is to include all four safe edges in the backbone. If each pair of vertices was connected via a safe edge, however, we could connect v with one of the terminal pairs and directly connect the other terminal pair with a safe edge. In this case we only used 3 edges. Intuitively, we were able to use one edge less by ignoring the connection between the two pairs of terminals. Since we assume our guess to be correct, we do not need to connect these terminals inside C as they will be connected via some paths outside of C. Thus, we want each set of vertices that are guessed to be pairwise connected via paths of safe edges inside C to form their own connected component when considering the graph induced by C in the solution.

if there is a safe edge between them in G. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_r\}$ with $R_i \subseteq Y$ be a partition of the vertices in Y according to which vertices are pairwise connected via paths of safe edges. Let $R'_i = \{y, y' \mid y \in R_i\}$ be the corresponding vertices in our constructed graph. As mentioned earlier, we need to consider the special case where one of the connecting components is a cyclic component with exactly two vertices in it as we need to ensure that there is at least one additional vertex that can be included in the cyclic component. In this case, we add two new vertices z, z' to the graph, connect each of them to all vertices v with $v \in C$, and define the set $R'_0 = \{z, z'\}$. We now solve Disjoint Connected Subgraphs where each set R'_i is one terminal set. Next, we show that if there is a set of disjoint connected subgraphs each connecting the vertices in one set R'_i, then this corresponds to a solution to both points. Note that we may again assume that each vertex in C can only fill the role of one vertex in R_i and hence selecting any vertex adjacent to y in the solution gives us a matching between the vertices in C and the vertices in Y (any vertex $y \in R_i$ needs to be connected to at least y' and hence such a neighbor exists). Moreover, since the solution is a set of disjoint connected subgraphs and we only included safe edges between vertices in C, we are also ensured that this solution corresponds to a set of connecting components as guessed. Conversely, if there is a set of disjoint subgraphs in C that exactly correspond to our set of guesses, then this is also a solution to the instance of Disjoint Connected Subgraphs if we additionally connect each pair of vertices y, y' to the respective vertex in C. Thus, we have found a way to check whether C can host a guessed set $\mathcal{P}$ of connecting components.

After checking which cliques could potentially host each set $\mathcal{P}$ of guessed connecting components, it remains to check whether all of these guesses can be fulfilled at the same time. To this end, we need to check whether all $q \leq p$ sets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_q$ can be hosted each by a distinct clique. We do this using the textbook $\mathcal{O}(nm)$-time algorithm for Maximum Bipartite Matching as follows. We build a bipartite graph with one vertex v_i for each set $\mathcal{P}_i$ and a vertex u_j for each connected component (clique) C_j in G'. There is an edge $\{v_i, u_j\}$ in the graph if and only if C_j can host $\mathcal{P}_i$. It then only remains to check whether there is a matching of size q as in this case each set $\mathcal{P}_i$ is matched to a distinct clique in G'.

For space reasons, we defer the analysis of the running time to the full version.

▷ Claim 4.4 (⋆). The algorithm runs in time $f(\ell) \cdot n^3$ for some computable function f. ◀

We mention that each cluster graph is also a co-graph. Whether the parameter distance to co-graphs also allows for an FPT-time algorithm remains an open question.

5 Conclusion

In this work, we started an investigation into the parameterized complexity of UFGC. Our main results are FPT-time algorithms for a below-upper-bound parameter, the treewidth, and the vertex-deletion distance to cluster graphs, respectively. Moreover, we give a fairly comprehensive dichotomy between parameters that allow for FPT-time algorithms and those that lead to W[1]-hardness (or in most cases even para-NP-hardness).

Nonetheless, several open questions remain. First, what is the status of the parameters that we were not able to resolve; for example does parameterization by the distance to cographs or interval graphs allow for an FPT-time algorithm? Second, is there an XP-time algorithm for UFGC parameterized by the clique-width. Third, it would be interesting to study the existence of polynomial kernels for the parameters that yield FPT-time algorithms. Unfortunately, some of these can be excluded due to the close relation to Hamiltonian Cycle. In particular, Hamiltonian Cycle (and hence also UFGC) does not admit a polynomial kernel with respect to the distance to outerplanar graphs unless coNP $\not\subseteq$ NP/poly [7]. Moreover, using the framework of AND-cross compositions, it is not hard to also exclude polynomial kernels for the parameters treedepth and bandwidth unless coNP $\not\subseteq$ NP/poly. However, this still leaves quite a few parameters ready to be investigated in the future.

Last but not least, UFGC should only be regarded as a first step towards generalizing Spanning Tree to more robust connectivity requirements. It is interesting to see whether (some of) our positive results can be lifted to the more general problem (p, q)-Flexible Graph Connectivity. Therein, the solution graph should still be q connected even if up to p unsafe edges fail. Also, one can study other problems in the setting where the edge set is partitioned into safe and unsafe edges from the lens of parameterized complexity; one possibility is (p, q)-Flexible (s, t)-Path [2].

References

1 David Adjiashvili, Felix Hommelsheim, and Moritz Mühlenthaler. Flexible graph connectivity. *Mathematical Programming*, 192(1):409–441, 2022.

2 David Adjiashvili, Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt. Fault-tolerant edge-disjoint $s-t$ paths - beyond uniform faults. In *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT '22)*, volume 227 of *LIPIcs*, pages 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

3 Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. NP-completeness of the hamiltonian cycle problem for bipartite graphs. *Journal of Information processing*, 3(2):73–76, 1980.

4 Jørgen Bang-Jensen, Manu Basavaraju, Kristine Vitting Klinkby, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms for survivable network design with uniform demands. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18)*, pages 2838–2850. SIAM, 2018.

5 Jørgen Bang-Jensen and Anders Yeo. The minimum spanning strong subdigraph problem is fixed parameter tractable. *Discrete Applied Mathematics*, 156(15):2924–2929, 2008.

6 Ishan Bansal, Joseph Cheriyan, Logan Grout, and Sharat Ibrahimpur. Improved approximation algorithms by generalizing the primal-dual method beyond uncrossable functions. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP '23)*, volume 261 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

7 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theoretical Computer Science*, 511:117–136, 2013.

8 Sylvia C. Boyd, Joseph Cheriyan, Arash Haddadan, and Sharat Ibrahimpur. Approximation algorithms for flexible graph connectivity. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '21)*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

9 Chandra Chekuri and Rhea Jain. Approximation algorithms for network design in non-uniform fault models. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP '23)*, volume 261 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

10 Ali Çivril. A new approximation algorithm for the minimum 2-edge-connected spanning subgraph problem. *Theoretical Computer Science*, 943:121–130, 2023.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

13 Andreas Emil Feldmann, Anish Mukherjee, and Erik Jan van Leeuwen. The parameterized complexity of the survivable network design problem. In *Proceedings of the 5th Symposium on Simplicity in Algorithms (SOSA '22)*, pages 37–56. SIAM, 2022.

14 Corinne Feremans, Martine Labbé, and Gilbert Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.

15 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.

16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Transactions on Algorithms*, 15(1):9:1–9:27, 2019.

17 Harold N. Gabow and Suzanne Gallagher. Iterated rounding algorithms for the smallest k-edge connected spanning subgraph. *SIAM Journal on Computing*, 41(1):61–103, 2012.

18 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

19 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980.

20 Woonghee Tim Huh. Finding 2-edge connected spanning subgraphs. *Operations Research Letters*, 32(3):212–216, 2004.

21 Christoph Hunkenschröder, Santosh S. Vempala, and Adrian Vetta. A 4/3-approximation algorithm for the minimum 2-edge connected subgraph problem. *ACM Transactions on Algorithms*, 15(4):55:1–55:28, 2019.

22 Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

23 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999.

24 Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009.

25 Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

26 András Sebö and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

27 Lawrence V Snyder, Maria P Scaparra, Mark S Daskin, and Richard L Church. Planning for disruptions in supply chain networks. In *Models, methods, and applications for innovative decision making*, pages 234–257. Informs, 2006.

28 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2000.

29 Ying Xu, Victor Olman, and Dong Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics*, 18(4):536–545, 2002.