A Brief Introduction to Parameterized Complexity

Ishan Bansal (ib332), Haripriya Pulyassary (hp297)

December 15, 2023

1 Introduction

Parameterized complexity adds a finer layer to the analysis of languages than classical complexity theory. Classically, metrics like computation time and space have been measured with respect to the input size of an instance and nothing more. Parameterized complexity on the other hand, considers additional parameters of the input instance allowing for a more fine-grained analysis. Many interesting problems in computer science have natural parameters defining the problem. For instance the vertex cover problem *"Is there a vertex cover of size k"?* has the size of the vertex cover k as a parameter which is independent from the size of the input graph. Similarly, the graph coloring problem *"Can the graph be colored by at most k colors"?* has the number of colors k as a natural parameter. As we will shortly see, the vertex cover problem has efficient parameterized algorithms while the situation for the graph coloring problem is hopeless (unless P=NP). There were examples of parameterized algorithms quite early on like Lenstra's algorithm for integer programming [15] and the disjoint paths algorithm by Robertson and Seymour [16]. However, the formalism with which it is studied today can be attributed to the early works of Downey and Fellows [10] in the late 1980s. In this report, we present a brief overview of the basic concepts in parameterized complexity including fixed-parameter tractability, slice-wise polynomial solvability, kernelization, the W-hierarchy and Turing machine equivalences. We end with a discussion on recent developments in the area of Steiner connectivity problems and an interesting open question therein.

2 Preliminaries

In this section, we introduce the formal definitions of parameterized complexity.

Definition 2.1. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite, fixed alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, we refer to k as the parameter.

A well-studied parameterized problem is the Vertex Cover problem. Given a graph $G = (V, E)$, a *vertex cover* is a set of vertices $S \subseteq V$ such that every edge has at least one endpoint in S. The Vertex Cover problem is the following: Given a graph G and *parameter* k, does there exist a vertex cover of size at most k? While we do not believe that there exists an algorithm for Vertex Cover running in time $poly(n, k)$ (unless $P = NP$), the following recursive algorithm solves Vertex Cover in $O(2^k n)$ time.

Algorithm 1 VertexCoverFPT(G, k)

```
if G has no edges then
    return True
end if
if k = 0 then
    return False
end if
uv: an edge of G.
if VertexCoverFPT(G − u, k − 1) or VertexCoverFPT(G − v, k − 1) then
    return True
end if
return False
```

We argue that the above algorithm is correct. The base cases are straightforward: if G has no edges, the empty set is a vertex cover, and hence we return True. If G has at least one edge, the size of a vertex cover is at least one – and hence, if $k = 0$ in this case, we return False. Otherwise, we are in the recursive case. Let uv be some edge in G. Note that any vertex cover S must contain u or v. Moreover, S is a vertex cover of G if and only if $S - u$ is a vertex cover of $G - u$, or $S - v$ is a vertex cover of $G - v$. So, (G, k) is a Yes-instance of Vertex Cover if and only if $(G - u, k - 1)$ is a Yes-instance of Vertex Cover, or $(G - v, k - 1)$ is a Yes-instance of Vertex Cover.

The runtime of VertexCoverFPT, $O(2^k n)$, is polynomial in the size of G, and hence we say it is "efficient". More formally, we say that VertexCoverFPT is a *fixed-parameter*

tractable (FPT) algorithm; just as polynomial-time algorithms are considered "efficient" in the standard setting, in parameterized complexity, FPT algorithms are considered to be the holy grail of efficiency.

Definition 2.2. A parameterized problem L is *fixed-parameter tractable (FPT)* if there exists a computable function f, constant c, and algorithm which correctly decides if $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$.

We will typically assume that the function f is computable and non-decreasing (indeed, this is without loss of generality as for every computable function $f : \mathbb{N} \to \mathbb{N}$, there exists a computable nondecreasing function $\bar{f}$ that is never smaller than f).

Another parameterized problem of interest is the Independent Set problem. Given a graph $G = (V, E)$, an *independent set* is a set $S \subseteq V$ such that for any pair of vertices $u, v \in S$, $uv \notin E$. The Independent Set problem asks the following question: given a graph G and parameter k, does there exist an independent set of size at least k? While we do not believe that this problem belongs to FPT, there exists an algorithm for Independent Set which runs in $O(k^2 n^k)$ time; we say that such problems belong to the class XP. More formally,

Definition 2.3. A parameterized problem is *slice-wise polynomial (XP)* if there exists computable functions f, g and an algorithm which correctly decides if $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^{g(k)}$.

Analogous to the traditional setting, we will find it helpful to use reductions to make meaningful statements regarding the relative difficulty of different parameterized problems. Instead of Karp reductions, we will work with *parameterized reductions*, which was first introduced by Downey and Fellows. One important property of such reductions is that if there is a parameterized reduction from A to B and B is in FPT, then A is also in FPT.

Definition 2.4. Let A, B be two parameterized problems. A *parameterized reduction* from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a yes-instance of A iff (x', k') is a yes-instance of B
2. $k' \leq g(k)$ for some computable function g
3. The running time is $f(k) \cdot |x|^{O(1)}$

3 Kernelization

When faced with the task of solving an NP-Hard problem, almost every practical computer implementation performs preprocessing steps or data reduction steps. The goal of such a subroutine is to quickly solve the easy parts of a problem instance and reduce it to its difficult core structure that can then be solved by slower exact algorithm. The language of parameterized complexity helps us in formalizing this concept and analyzing the effectiveness of such preprocessing subroutines. This cannot be done in the context of classical complexity since if the bit size of an instance of an NP-Hard problem could be reduced by even just one bit, then this would imply that P=NP.

3.1 Formal Definitions

We begin with some formal definitions of data reductions and kernelizations.

Definition 3.1. A *data reduction rule* for a parameterized problem L is a function $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance (I, k) of L to an instance (I', k') of L such that ϕ is computable in polynomial time in $|I|$ and k.

We will be interested in reduction rules that are *safe* which means that $(I, k) \in L$ if and only if $(I', k') \in L$. Unless otherwise stated, all reduction rules we discuss will be required to be safe. We measure the size of a data reduction rule as follows.

Definition 3.2. The *output size* of a data reduction rule ϕ is a function $\text{size}_\phi : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ defined as

$$\text{size}_\phi(k) = \sup\{|I'| + k' : (I', k') = \phi(I, k),\ I \in \Sigma^*\}$$

In other words, we consider all possible instances of the problem L with a fixed parameter k and measure the supremum of the sizes of the resulting instances after applying the data reduction rule ϕ. A kernelization algorithm, or simply kernel is a data reduction rule whose output size is bounded by a function of the parameter k.

Definition 3.3. A *kernelization algorithm*, or simply a *kernel*, for a parameterized problem L is a safe data reduction rule ϕ such that $\text{size}_\phi(k) \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

The above definition of kernelization captures the efficiency of the algorithm in terms of the amount of data reduced or the function g. While practically, the running time may also be of concern, in theory the only requirement is that the algorithm runs in polynomial time. This definition of kernelization relates back to the notion of fixed parameter tractability in a very natural way. The following lemma is due to Cai et al. [5].

Lemma 3.4 (Cai et al. [5])**.** *A parameterized problem L is in FPT if and only if it admits a kernelization algorithm.*

Proof. First, if the problem L admits a kernelization algorithm ϕ, then given an instance (I, k), we can apply the algorithm $\phi(I, k)$ to obtain an equivalent instance (I', k') in polynomial time such that $|I'| + k' \leq g(k)$ for some computable function g. But then, we can apply a brute force search on all instances of size at most $g(k)$ if required to decide the instance (I', k'). This provides an FPT algorithm to decide the instance (I, k).

For the reverse direction, assume that the parameterized problem L is in FPT. Hence, there is an algorithm $\mathcal{A}$ that decides if $(I, k) \in L$ with running time $f(k) \cdot |I|^c$ for some constant c and some computable function f. We design the kernelization algorithm for L as follows. Run the algorithm $\mathcal{A}$ on the input (I, k) for $|I|^{c+1}$ steps. If the algorithm stops, we have decided (I, k) and hence shrunk its size to one bit (yes or no). If not, then we can conclude that $|I|^{c+1} \leq f(k) \cdot |I|^c$ and so $|I| \leq f(k)$. Hence, the kernelization algorithm can output the same instance (I, k) and this has size at most $f(k) + k$. □

3.2 Kernel for the Vertex Cover Problem

In this subsection, provide a kernelization algorithm for the vertex cover problem discussed earlier. Here, we are given a graph G and a positive integer k as input, and the goal is to decide if there exists a vertex cover of size at most k. We exhibit two reduction rules, the repeated application of which would lead to a kernelization algorithm. This kernel was shown by Buss and Goldsmith [4].

The first reduction rule is almost trivial. If the graph G has an isolated vertex v, then, this vertex does not cover any edge and so we can safely remove it from the graph G.

Reduction Rule VC.1. If G contains an isolated vertex v, delete v from G. The new instance is $(G - v, k)$.

The second rule is based on the following observation. Suppose there exists a vertex v in G of degree strictly larger than k, then the vertex v must lie in any vertex cover of size at most k. This is because, all the edges incident to the vertex v must be covered and there are at least $k + 1$ of them. Hence, the vertex cover cannot always include the other end-point of these edges and must therefore include the vertex v. Thus our second rule is:

Reduction Rule VC.2. If G contains a vertex v of degree at least $k + 1$, then delete v and its incident edges and reduce the parameter k by 1. The new instance is $(G - v, k - 1)$.

Observe that both reduction rules can be applied in linear time and can be applied at most n times where n is the number of vertices in the original graph G. Hence in polynomial time, one can repeatedly apply reduction rules VC.1 and VC.2 in any desired order until none of the rules are applicable anymore. We argue that in such a scenario, the size of the graph G can be bounded by a function of k if the original instance was a yes instance.

Lemma 3.5. *Let (G, k) be a yes-instance of the vertex cover problem. Let (G', k') be the instance obtained after exhaustively applying reduction rules VC.1 and VC.2. Then, $|V(G')| \leq k'^2 + k'$ and $|E(G')| \leq k'^2$.*

Proof. Note that both reduction rules VC.1 and VC.2 are safe and hence (G', k') is also a yes-instance. Furthermore, $k' \leq k$ since these reduction rules do not increase the parameter. Let S be a vertex cover of G' of size at most $k' \leq k$. Since we cannot apply reduction rule VC.1 to G', it does not have any isolated vertices. Hence, every vertex of G' is adjacent to some vertex of S. Since we cannot apply reduction rule VC.2 to G', we know that every vertex in S has degree at most $k' \leq k$. Hence the total number of vertices in the graph G' can be bounded by $|V(G')| \leq k'^2 + k' \leq k^2 + k$. Furthermore, every edge in the graph G' is incident to a vertex in S by the definition of a vertex cover. Hence, the number of edges in the graph G' can be bounded bu $|E(G')| \leq k'^2 \leq k^2$. □

With the above lemma in place, we can obtain our final reduction rule that explicitly

bounds the size of the kernel.

Reduction Rule VC.3. Let (G, k) be an instance such that reduction rules VC.1 and VC.2 are not applicable. If G has more than $k^2 + k$ vertices or more than k^2 edges, then conclude that (G, k) is a no-instance.

Putting together the above three reduction rules provides us with a kernelization algorithm where the size of the final graph is bounded by a function of k. We explicitly state this in the following theorem.

Theorem 3.6 (Buss, Goldsmith [4])**.** *The vertex cover problem admits a kernel with $O(k^2)$ vertices and $O(k^2)$ edges.*

We would like to point out that the above result is in some sense tight by highlighting a recent result in the context of lower bounds for kernelization. Due to the limited scope of this project, we cannot provide proofs of the following theorems, but note that finding lower bounds for kernelization algorithms is a very active area of research.

In a breakthrough result, Dell and Marx [9] proved the following theorem

Theorem 3.7 (Dell and Marx [9])**.** *For any $\epsilon > 0$, the vertex cover problem does not admit a kernel of size $O(k^{2-\epsilon})$, unless* NP $\subseteq$ coNP/poly*.*

The proof of this result uses heavy machinery developed in the context of kernelizations like compositions and distillations. At a very high level, the authors show that one can create a vertex cover instance by considering a suitably defined *OR*-composition of a number of multi-colored biclique problem instances. The multi-colored biclique problem is known to be NP-Hard. They then apply the following theorem.

Theorem 3.8. *Assume that an* NP-Hard *language L admits a weak cross-composition of dimension d into a parameterized language Q. Suppose Q admits a kernel of size $O(k^{d-\epsilon})$ for some $\epsilon > 0$, then* NP $\subseteq$ coNP/poly*.*

We refer the reader to Chapter 15 in the book by Cyagn et al.[8] for a detailed exposition of these results and ideas. Note that if the hypothesis NP $\subseteq$ coNP/poly is true then, the polynomial hierarchy collapses to the third level as shown by Yap [17].

4 The W-hierarchy

The W-hierarchy was introduced by Downey and Fellows, as an attempt to capture the exact complexity of various hard parameterized problems. This hierarchy is defined using reductions to certain classes of Boolean circuits. Recall that a Boolean circuit is a directed acyclic graph every node of indegree 0 is an *input node*, every node of indegree 1 is a *negation node*, and every other node is an *AND*-node or an *OR*-node. Furthermore, exactly one of the nodes with outdegree 0 is the *output node*. The *depth* of a circuit is the maximum length of a path from an input node to the output node. We say that a node is "large" if it has an indegree strictly larger than 2. The *weft* of a boolean circuit is the maximum number of large nodes on any path from an input node to the output node.

An assignment of 0-1 values to the input nodes determines the value of all other nodes in the circuit. Such an assignment is *satisfiable* if the output gate has a value of 1. We say that the *weight* of an assignment is the number of input gates receiving a value of 1. We are now ready to define the following parameterized circuit satisfiability problem, which will be crucial when defining the W-hierarchy.

Weighted Circuit Satisfiability (WCS): Given circuit C and an integer k, does C have a satisfying assignment of weight exactly k?

If $\mathcal{C}$ is a class of circuits, then we define WCS[$\mathcal{C}$] to be the restriction of the problem where the input circuit C belongs to $\mathcal{C}$. We are particularly interested in $\mathcal{C}_{t,d}$, the class of circuits of weft at most t and depth at most d.

Definition 4.1 (**W-hierarchy**)**.** A parameterized problem P belongs to **W[t]** if there is a parameterized reduction from P to WCS[$\mathcal{C}_{t,d}$] for some $d \geq 1$.

Notice that FPT=W[0] and W[i] $\subseteq$ W[j] for all $i \leq j$. Furthermore, we say a problem P is W[t]-hard if, for every $P' \in$ W[t], there exists a parameterized reduction from P' to P. An example of a W[1]-complete problem is the Independent Set problem discussed earlier. The proof that Independent Set is W[1]-complete is non-trivial and hence is omitted. We do, however, provide an argument showing that Independent Set is in W[1].

Let (G, k) be an instance of the Independent Set problem. We construct a boolean

circuit C in the following manner: Each node $v \in V$ corresponds to an input node of the circuit, and is connected to a negation node. For every edge $uv \in E$, the negation nodes corresponding to u and v are connected to an OR node; and all OR nodes (corresponding to the edges) are connected to a single AND node, which is also the output node. Note that this circuit has exactly one large node (the output node), and hence has a weft of 1. An example of an INDEPENDENT SET instance and its corresponding boolean circuit is given in Figure 1.

We claim that an assignment in this circuit is satisfiable if and only if it corresponds to an independent set in G. If we have an independent set S, we can assign a value of 1 to the input node corresponding to each $v \in S$; this is a satisfiable assignment as for each OR node (corresponding to an edge), at least one of the inputs will be 1. Moreover, the weight of this assignment is precisely the size of the independent set. We now prove the other direction. Suppose we have a satisfiable assignment, and assume to arrive at a contradiction that $uv \in E$ for some u, v which have been assigned a value of 1. Then, the OR-gate corresponding to uv receives zeros as both of its input, and hence passes a value of 0 to the output AND node – which is a contradiction. Every satisfiable assignment corresponds to an independent set; moreover, the weight of the assignment is the size of the independent set. Thus, INDEPENDENT SET reduces to finding a satisfiable assignment of weight at most k in a weft-1 circuit.

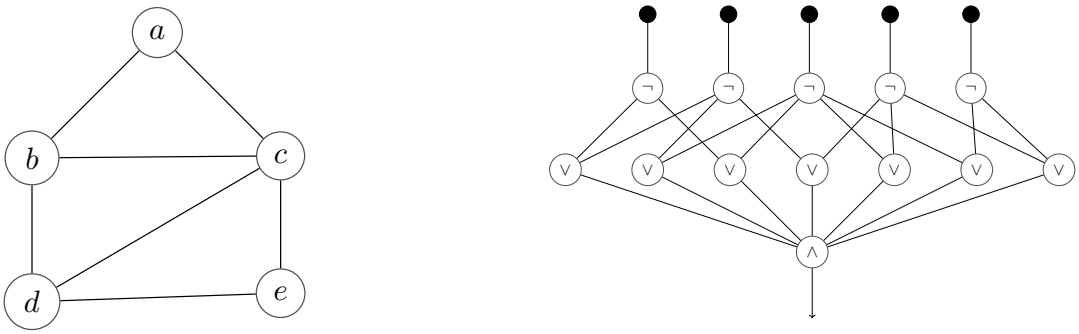

Figure 1: An INDEPENDENT SET instance

An interesting observation about the W-hierarchy is that NP-COMPLETE problems, which are equivalent to each other with respect to polynomial-time reductions, occupy different classes in the W-hierarchy. For instance, the SET COVER problem (wherein one is given a universe U, collection of sets $\mathcal{S}$, parameter k, and must decide *does there exist* $\mathcal{S}' \subseteq \mathcal{S}$ *such that* $\cup_{S \in \mathcal{S}'} S = U$ *and* $|\mathcal{S}'| \leq k$*?*) is NP-COMPLETE; hence, it is equivalent, with respect to

polynomial-time reductions, to the INDEPENDENT SET problem. However, in terms of the W-hierarchy, SET COVER is $W[2]$-complete and hence is in a different parameterized complexity class than INDEPENDENT SET.

The definition of the W-hierarchy, and particularly the class W[1] is in terms of boolean circuits. Cai et al [5] studied the connection between W[1] and Turing machines. Informally, when answering the $P \neq NP$ problem, one must decide in deterministic polynomial time if a non-deterministic Turing machine (NDTM) has an accepting path; the parameterized analogue is to decide in FPT time if an NDTM has a computation path reaching an accepting state in at most k steps. Formally, we define the Short Turing Machine Acceptance problem as follows.

SHORT TURING MACHINE ACCEPTANCE: Given an NDTM M, a string x, and an integer k; the task is to decide if M with input x has a computation path reaching an accepting state in at most k steps.

The SHORT TURING MACHINE ACCEPTANCE problem is W[1]-COMPLETE. It is conjectured that FPT≠W[1]. As SHORT TURING MACHINE ACCEPTANCE problem is W[1]-COMPLETE, and it seems unlikely that there exists an FPT algorithm that can decide if M with input x has a computation path reaching an accepting state in at most k steps, it seems more likely that FPT≠W[1].

5 Steiner Connectivity Problems

In this section we discuss some network design problems in the context of parameterized complexity. In particular we will discuss Steiner connectivity problems. The general goal of network design is to find a cheap subgraph of a given input graph that satisfies some pre-defined connectivity requirements. For instance, in the well-known minimum spanning tree problem, we are given a graph $G = (V, E)$ with costs on the edges, and the goal is to find a cheapest subgraph (V, F) which is connected. In Steiner connectivity problems, the connectivity requirements are only between a given subset of the node set called *terminals*. Other nodes are called *Steiner nodes*, named after the Swiss mathematician Jakob Steiner.

The analog of the minimum spanning tree problem in the Steiner setting is the Steiner tree problem where the input is a graph $G = (V, E)$ with edge costs and a set of terminals $T \subseteq V$. The goal is to find a cheapest subgraph $G' = (V', E')$ such that all terminals are included, $T \subseteq V'$, and the graph G' is connected. While the minimum spanning tree problem is polytime solvable, the Steiner tree problem is known to be NP-Hard. We provide a short survey of some known results in the parameterized analysis of Steiner connectivity problems and end with an interesting open question. We will also try to provide sketches of proofs where possible. We restrict our attention to Steiner connectivity problems where every edge has unit cost.

We mention two natural parameters that we will consider in our discussions. First, is the solution size that we denote by ℓ. The parameterized problem here is *"Does there exists a feasible solution with at most ℓ edges"* where a feasible solution depends on the specific problem in hand. The second parameter is the number of terminals in the input instance, $|T|$. We denote this by k. The parameterized problem here is *"Find the cheapest feasible solution given k terminals"*. Note that we can always assume $k \in O(\ell)$ since to even just connect k terminals, we require at least $k-1$ edges. Hence, a hardness result with respect to parameter ℓ will imply a hardness result with respect to parameter k. Similarly, an algorithm with respect to parameter k will imply an algorithm with respect to parameter ℓ.

The first result we discuss is an old result by Dreyfus and Wagner [11] showing that the Steiner tree problem is in FPT with respect to the parameter k (hence also parameter ℓ) in both directed and undirected graphs.

Theorem 5.1 (Dreyfus, Wagner [11])**.** *The Steiner Tree problem can be solved in both directed and undirected graphs in time* $3^k n^{O(1)}$.

Proof. The insight here is to use a dynamic programming algorithm and identify that connected subtrees of the optimal Steiner tree must be optimal Steiner trees themselves on a restricted set of terminals. Define the states of the dynamic program to be $H[T', v]$ where T' is a subset of the terminals and $v \in V$ is some vertex that will act as a root. $H[T', v]$ denotes the optimal Steiner Tree connecting every terminal in T' with v as a root. The dynamic program can be initialized by observing that if $|T'| = 1$, then $H[T', v]$ is simply the shortest path from v to the node in T'. The recursion for the program can be obtained by observing

that for any subset of terminals T' and root v, we can start from v and follow the nodes in $H[T', v]$ until we arrive at the first node u of degree at least three (this could be v itself). Then, this node creates multiple branches of the tree $H[T', v]$ and each branch has a subset of the terminals T' in it. But then, we can use the information from states previously computed for $H[T'', u]$ to obtain the optimal tree $H[T', v]$. Formally,

$$H[T', v] = \min_{\substack{u \in V \setminus T \\ T'' \subsetneq T'}} H[T'', u] + H[T' \setminus T'', u] + dist(v, u)$$

Note that, we are abusing notations by using $H[T', v]$ to denote both the optimal tree and its cost. Additionally, we take $u \in V \setminus T$ in the minimum above since we can assume that terminals in the input graph have degree exactly one by essentially hanging them off the original terminals. The above recursion provides an exact dynamic programming algorithm to solve the Steiner tree problem and its run-time can be calculated to be $3^k n^{O(1)}$. □

To be connected in an undirected graph is to have a path between every pair of vertices. In the context of directed graphs, the equivalent notion is called strongly connectedness. While the Steiner tree problem is in FPT for undirected (and directed) graphs, the Steiner strongly connected subgraph problem has been shown to be W[1]-Hard with respect the the parameter ℓ (hence also parameter k).

Theorem 5.2 (Guo, Niedermeier, Suchy [14])**.** *The Steiner strongly connected subgraph problem is* W[1]-Hard *with respect to the parameter* ℓ*.*

Proof. We present a brief sketch of the proof when ℓ is the number of vertices in a solution and not edges for simplicity. The idea is to provide a parameterized reduction from the W[1]-Hard problem Multicolored Clique. Here the input is a graph G with a partition of the vertices into t color classes $V_1, \ldots, V_t$. The goal is to find a clique containing one vertex from each color class. The intuitive idea of the reduction is to create vertices for every edge in the graph G and to force a feasible strongly connected solution to pick up one vertex from each color class and $\binom{t}{2}$ edges connecting these vertices. We then set the value of ℓ accordingly so that a strongly connected subgraph with at most ℓ vertices implies the existence of a multicolored clique in the original graph. We refer the reader to the original paper [14] for the full details. □

While on general directed graphs, the Steiner strongly connected subgraph problem is W[1]-HARD, it has been shown that if one restricts their attention to bidirected instances, then the problem becomes fixed parameter tractable with respect to parameter k (hence also parameter ℓ). A bidirected instance is one where if an edge exists, then the edge in it reverse direction also exists.

Theorem 5.3 (Chitnis, Feldmann, Manurangsi [7]). *The Steiner strongly connected subgraph problem on bidirected graphs can be solved in time* $4^{O(k^2)}n^{O(1)}$.

Proof. The algorithm here also proceeds via dynamic programming but is far more involved than the Dreyfus-Wagner algorithm for Steiner trees. A key observation here is that an optimal solution can be decomposed into smaller strongly connected components of treewidth one (called poly-trees), whose union gives the whole solution. These smaller strongly connected components contain a subset of the terminal set T in specific patterns. All such patterns can be enumerated in FPT time $2^{O(k^2)}$, and once a pattern is fixed, one can apply a previous result by Feldmann and Marx [12] to compute optimal poly-trees. There they show that if the optimal solution of a Steiner strongly connected subgraph problem has constant treewidth, then it can be found in FPT time. Now, one can apply a dynamic programming technique similar to the Dreyfus-Wagner result. The difference being that in the Steiner tree problem, the DP is trying to attach trees together, while here we will attach poly-trees together. □

Moving back to undirected graphs, if we take a strongly connected directed graph and undirect every edge or treat every edge as being undirected, then we obtain what is called a 2-edge connected graph. Formally, a graph is 2-edge connected if there exist two edge-disjoint paths between every pair of nodes u, v. A cycle is the simplest 2-edge connected subgraph. The Steiner cycle problem asks for a cheapest subgraph which is a cycle and contains all the terminals. The following result shows that the Steiner cycle problem is in FPT with respect to parameter k (hence also parameter ℓ). The techniques used to prove this result are more algebraic.

Theorem 5.4 (Bjorklund, Husfeldt, Taslman [3]). *The Steiner cycle problem can be solved in time* $2^k n^{O(1)}$ *via a randomized algorithm.*

Proof. The idea here is to assign a variable to each edge in the graph and compute a polynomial over some large field of characteristic 2 that evaluates to a non-zero if and only if

there exists a cycle through all the terminals of a specified length h. Then, one can randomly assign values to the edge variables and apply the Shwartz-Zippel lemma to decide if there exists such a cycle. The particular polynomial is computed via a dynamic program. In particular, it computes a sum of all closed walks of length h starting and ending at a chosen terminal t and passing through all other terminals. If this walk has a sub-cycle, then going around this cycle in different orientations, provides two different ways to obtain the same polynomial term. Since the field has characteristic two, these two terms cancel out. Hence, the only closed walks that survive are cycles. We can run this algorithm for different guesses of $h = 1, \ldots, n$ to obtain the smallest h that provides a Steiner cycle. Note that the above mechanism only detects the length of the optimal solution but does not find it. To actually find the optimal solution, one can run this idea again and again after guessing the first edge of the Steiner cycle and then the second edge and so on. □

Generalizing the Steiner cycle problem, we obtain the Steiner 2-edge connected problem. Here, the goal is to compute a cheapest subgraph $H = (V', E')$ of the input graph G such that H is 2-edge connected and contains all the terminals, $T \subseteq V'$. The following result places this problem in FPT with respect to the parameter ℓ, but not with respect to the parameter k!

Theorem 5.5 (Feldmann, Mukherjee, Leeuwen [13])**.** *The Steiner 2-edge connected problem can be solved in time* $2^{O(\ell \log \ell)} n^{O(1)}$ *time.*

Proof. The proof relies on a common color coding technique that is useful when the parameter is the solution size, along with a neat decomposition of optimal solutions. Using an earlier result by Chekuri and Korula [6] to show that an optimal solution can be decomposed into edge-disjoint trees whose leaves are terminals. To figure out what the correct partition of terminals and edges are into these edge-disjoint trees, one can randomly color the edges and vertices of the input graph so that with probability at least $2^{-O(\ell \log \ell)}$, each of the optimal edge-disjoint trees that the optimal solution decomposed into lies in a different color class. One can then apply FPT algorithms for Steiner tree to compute the edge-disjoint trees and take their union to obtain the final solution. The color coding technique is borrowed from the work of Alon, Yuster and Zwick [1] and can be derandomized. □

The above result does not say anything about the status of the Steiner 2-edge connected

subgraph problem with respect to the parameter k. Recently, we showed that the problem lies in XP.

Theorem 5.6 (Bansal, Cheriyan, Grout, Ibrahimpur [2])**.** *The Steiner 2-edge connected subgraph problem can be solved in time $n^{O(k)}$ via a randomized algorithm where n is the number of vertices in the input graph.*

Proof. The idea here is to utilize the ear decomposition of 2-edge connected subgraphs. Essentially what this says is that any 2-edge connected subgraph can be decomposed into cycles (ears). Hence, a natural algorithm is to guess the terminals on each ear of the decomposition and use the earlier result of Bjorklund et al. [3] to find optimal Steiner cycles passing through these terminals. However, one also needs to guess all nodes of the optimal solution of degree at least three to paste these cycles back together. The issue now is that the number of ears in the optimal solution and hence the number of high-degree nodes could be as large as $\Omega(n)$. To overcome this, one can consider the maximal 2-node connected subgraphs (blocks) of the optimal solution and the 2 edge-disjoint paths that connect these blocks together. Optimal 2-node connected subgraphs only have $O(k)$ high degree nodes and this solves the issue. Thus the algorithm essentially guesses all blocks of the optimal solution and then computes these blocks using the FPT algorithm for the Steiner cycle problem, and then pastes these blocks together using 2 edge-disjoint paths. □

5.1 Open Questions

To conclude our report, we mention some interesting open questions that arise naturally out of our previous discussions on Steiner connectivity problems. The first is to do with the parameterized complexity of the Steiner 2-edge connected subgraph problem. Feldmann et al. [13] showed that the problem is in FPT with respect to parameter ℓ and Bansal et al. [2] showed that the problem is in XP with respect to the parameter k. This begs the following question:

Open Question 1: Is the Steiner 2-edge connected subgraph problem in FPT or is it W[1]-Hard?

Note that all the results we have discussed are in the setting where all edge costs are unit. What happens when the edge costs are allowed to be arbitrary? Here, the Steiner tree problem is still in FPT as shown by Dreyfus and Wagner [11]. Bansal et al. [2] showed that the Steiner cycle problem admits a fully polynomial time approximation scheme (FPTAS) in FPT time and the Steiner 2-edge connected subgraph problem admits an FPTAS in XP time. But the question of exact algorithms remains open.

Open Question 2: Is the Steiner cycle problem with arbitrary edge costs in FPT?

Finally, we consider parameterized approximation algorithms for the Steiner 2-edge connected problem. It is well known that a polynomial time 2-approximation algorithm can be obtained for this problem via Jain's iterative rounding, for example [**?**]. But perhaps one can do better given FPT time.

Open Question 3: Is there an approximation algorithm for the Steiner 2-edge-connected subgraph problem that runs in time $f(k)n^{O(1)}$ that obtains an approximation factor smaller than 2?

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[2] Ishan Bansal, Joe Cheriyan, Logan Grout, and Sharat Ibrahimpur. Algorithms for 2-connected network design and flexible steiner trees with a constant number of terminals. *arXiv preprint arXiv:2206.11807*, 2022.

[3] Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on discrete algorithms*, pages 1747–1753. SIAM, 2012.

[4] Jonathan F Buss and Judy Goldsmith. Nondeterminism within pˆ. *SIAM Journal on Computing*, 22(3):560–572, 1993.

[5] Liming Cai, Jianer Chen, Rodney G Downey, and Michael R Fellows. On the parameterized complexity of short computation and factorization. *Archive for Mathematical Logic*, 36(4-5):321–337, 1997.

[6] Chandra Chekuri and Nitish Korula. A graph reduction step preserving element-connectivity and packing steiner trees and forests. *SIAM Journal on Discrete Mathematics*, 28(2):577–597, 2014.

[7] Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected steiner network problems. *ACM Transactions on Algorithms (TALG)*, 17(2):1–68, 2021.

[8] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

[9] Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 68–81. SIAM, 2012.

[10] Rod G Downey and Michael R Fellows. *Fixed-parameter tractability and completeness*. University of Victoria, Department of Computer Science, 1991.

[11] Stuart E Dreyfus and Robert A Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[12] Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed steiner network problems. *ACM Transactions on Computation Theory*, 2017.

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

[14] Jiong Guo, Rolf Niedermeier, and Ondřej Suchỳ. Parameterized complexity of arc-weighted directed steiner problems. *SIAM Journal on Discrete Mathematics*, 25(2):583–599, 2011.

[15] Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983.

[16] Neil Robertson and Paul D Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

[17] Chee K Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical computer science*, 26(3):287–300, 1983.