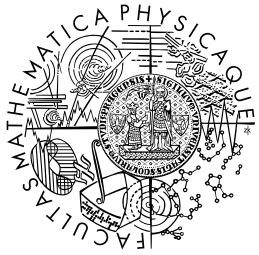

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER'S THESIS

Martin Koreček

Parameterized Approximations of Directed Steiner Networks

Department of Applied Mathematics

Supervisor of the Master's thesis: doc. Andreas Emil Feldmann, Dr.
Study programme: Computer Science
Study branch: Discrete Models and Algorithms

Prague 2023

I declare that I carried out this Master's thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor Andreas Emil Feldmann for his guidance and support in writing this thesis. I am sincerely grateful for the opportunity to benefit from his expertise.

Title: Parameterized Approximations of Directed Steiner Networks

Author: Martin Koreček

Department: Department of Applied Mathematics

Supervisor: doc. Andreas Emil Feldmann, Dr., Department of Applied Mathematics

Abstract: An instance of the Directed Steiner Network (DSN) problem consists of a directed graph G with edge costs, and k so called "terminal" vertex pairs. The task is to find a minimum-cost subgraph of G in which all terminal pairs are connected by a path. This generalizes several NP-hard problems.

The terminal pairs induce the so called "pattern graph", a digraph on a subset of vertices of G. We investigate the DSN problem restricted to certain classes of pattern graphs. It has been shown that the optimum may be found for certain classes in FPT time parameterized by k, and that this is impossible for all other classes of graphs, assuming FPT $\neq$ W[1].

This leads to the question whether the hard classes may be approximated in FPT time. We prove that no FPT approximation scheme may exist for any of the W[1]-hard classes, based on a stronger hypothesis, the Gap-ETH. We then give FPT algorithms with constant approximation guarantees for special classes of pattern graphs.

Keywords: parameterized algorithms, Steiner network, approximation, Fixed-Parameter Tractable

Contents

Introduction

In this thesis, we deal with one of the most general Steiner problems:

Definition 1. *An instance Δ of the* Directed Steiner Network *(DSN) problem consists of*

- *a directed graph $G = (V_G, E_G)$,*
- *a non-negative edge-cost function $c : E_G \to \mathbb{R}^+$ and*
- *k demand pairs $(s_1, t_1), \ldots, (s_k, t_k) \in V_G^2$, also called terminal pairs.*

The goal is to compute a minimum-cost subgraph of G containing a directed $s_i \to t_i$ path for each $1 \leq i \leq k$.

The demand pairs define a digraph H on the vertex set $\{s_1, t_1, \ldots, s_k, t_k\}$ with arcs $\{s_1t_1, \ldots, s_kt_k\}$. We call this the ***pattern graph*** *of Δ and the whole instance can be equivalently defined as the triplet $\Delta = (G, H, c)$.*

Such problems arise when designing networks given an underlying structure, where connections between nodes need to be paid for. Immediate applications arise in network design [1] and connectivity [2, 3]. DSN generalizes several combinatorial problems: the polynomially solvable shortest path and minimum spanning tree problems in undirected graphs are generalized by the Steiner Tree, one of the best known NP-hard optimization problems, listed in Richard Karp's seminal paper [4] among 20 other problems. The DSN generalizes this idea by considering directed graphs and optimizing for general connectivity between the graph's nodes. This problem has seen much algorithmic progress in recent years, as we will discuss.

The well-studied Directed Steiner Tree (DST) problem is a special case of DSN, where there is one "root" terminal and the output network is required to connect the root to every other terminal. The DST is NP-hard (by a trivial reduction from Steiner Tree), and parameterized by the pattern graph's number of edges k, it is FPT by the Dreyfus and Wagner algorithm [5].

Another important special case is the Strongly Connected Steiner Subgraph (SCSS) problem, where k terminal vertices of a directed graph are chosen and we are interested in the minimum-cost subgraph containing a path from any terminal to any other terminal. In other words, the SCSS corresponds to the DSN where the pattern graph is a directed cycle. This problem is not known to be FPT any longer, and it is in fact proven to be W[1]-hard [6].

Special cases like these motivate the following definition.

Definition 2. *For a class $\mathcal{H}$ of directed graphs, the $\mathcal{H}$-DSN is the special case of DSN where the pattern graph H is required to be in $\mathcal{H}$.*

This approach only restricts the pattern graphs, with no further requirements on the large graph G. Restricting G also leads to intriguing problems: assuming e.g. the graph to contain the vu edge whenever it contains uv yields interesting algorithmic results [7]. The approaches may also be combined, as done by [8], where a complete characterization of the parameterized complexity of $\mathcal{H}$-DSN is given for planar graphs (assuming ETH).

We will mainly build on [6], where a dichotomy is proven for the complexity of $\mathcal{H}$-DSN problems. We formalize their result in Section 1.1. Informally speaking,

they introduce caterpillar graphs: a λ-caterpillar is formed by a path of length λ, with each of its nodes further connected to an arbitrary number of leaf nodes, as illustrated in Figure 1. They further define $\mathcal{C}^*_{\lambda,\delta}$ as the class of λ'-caterpillars for $\lambda' \leq \lambda$, to which at most δ arbitrary edges may be added (in fact, $\mathcal{C}^*_{\lambda,\delta}$ is the closure of such class under transitive equivalence and the reversal of edge directions).

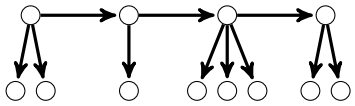

Figure 1: A 4-caterpillar graph.

The statement *informally* says the following, which we state precisely in Theorem 1.

For a class of pattern graphs $\mathcal{H}$,

- *either there are constants λ, δ such that $\mathcal{H} \subseteq \mathcal{C}^*_{\lambda,\delta}$, in which case there is an FPT algorithm for $\mathcal{H}$-DSN, parameterized by k,*
- *or, if there are no such constants λ and δ, $\mathcal{H}$-DSN is $W[1]$-hard (for parameter k), and therefore unlikely to be solvable in FPT time.*

Our results

Given the full characterization of the FPT-computability of $\mathcal{H}$-DSN provided by Theorem 1, we are curious about the $W[1]$-hard cases. Since it is likely impossible to solve such problems in FPT time, we inquire about approximation. Can we find FPT algorithms with favourable approximation bounds for certain $W[1]$-hard classes? Or can we even get an approximation scheme for some of them?

The ultimate challenge would be finding an approximation analogue to Theorem 1: a statement that would, for every recursively enumerable class of directed graphs $\mathcal{H}$, provide an FPT approximation algorithm for $\mathcal{H}$-DSN, and a matching approximation lower bound, based on plausible conjectures. We don't manage to fully reach this goal, but we make progress that allows us to answer several questions in this direction.

In Chapter 2, and Theorem 11 in particular, we prove that only constant-factor approximation is possible in FPT time, for any of these cases, if Gap-ETH holds (we will introduce this conjecture shortly). Theorem 11 essentially extends Theorem 1 by showing, under the assumption of Gap-ETH, the following.

If there are no constants λ, δ such that $\mathcal{H}$ is a subclass of $\mathcal{C}^*_{\lambda,\delta}$, there is no $f(k) \cdot n^{\mathcal{O}(1)}$-time $(\frac{4}{3} - \epsilon)$-approximation algorithm for $\mathcal{H}$-DSN, for any $\epsilon > 0$ and any function f.

In Chapters 3 and 4, we will design constant-factor FPT approximation algorithms for several $W[1]$-hard cases of $\mathcal{H}$-DSN. As a prelude, in Section 3.1, we design an FPT 2-approximation algorithm for $\mathcal{H}$-DSN where $\mathcal{H}$ is the class of directed paths. While being the simplest case we approximate in this thesis, the factor 2 is optimal under Gap-ETH, and no constant-factor approximation is possible in polynomial time, unless P = NP.

In Section 3.2, we extend this for $\mathcal{H}$ being the class of patterns composed of several directed paths (even more, only the condensation graph needs to be obtained this way). This already covers many classes of patterns, but leaves significant gaps. In particular, such algorithm doesn't cover the subclasses of $\mathcal{C}^*_{\lambda,\delta}$.

In Chapter 4, we cover and extend the cases of $\mathcal{H}$-DSN currently known to be constant-factor approximable in FPT, such as the SCSS, by introducing a quite general family of patterns which we call (χ, π)-Cat: the class of directed graphs whose condensation is the union of at most χ caterpillars and at most π paths, of arbitrary length. We show how to approximate the DSN restricted to such patterns, in Theorem 17. All of the approximation algorithms we give have optimal approximation ratios under certain complexity assumptions.

We discuss the obstacles to discovering the full FPT approximation complexity landscape of $\mathcal{H}$-DSN, and directions to help reach it, in Section 4.2. In particular, we introduce conjectures for certain pattern graph classes which we believe are likely to hold.

1. Preliminaries

Besides concepts elementary for theoretical computer science, such as graph, NP-hardness, decidability and Turing enumerability, we expect familiarity with

- parameterized algorithms: classes FPT, XP, the W-hierarchy and the Exponential Time Hypothesis (ETH) as treated e.g. by the book [9],
- approximation algorithms, approximation schemes (see e.g. [10]).

Our inapproximability results require the Gap-ETH hypothesis. For completeness, we state it here. We will not build on it directly, but we use its implications that we state throughout the thesis.

For a CNF formula Φ with m clauses, let $\mathrm{val}(\Phi)$ denote the maximum possible number of satisfied clauses divided by m.

Conjecture 1. *(Gap-ETH hypothesis) There exist constants ϵ, δ such that there is no $\mathcal{O}(2^{\delta n})$-time algorithm that, given a 3-CNF formula Φ with n variables, distinguishes between these two cases*[1]*:*

- $val(\Phi) = 1$,
- $val(\Phi) < 1 - \epsilon$.

The standard version of ETH implies (via the PCP theorem [11][12][13]) a weaker version of the Gap-ETH. It is open whether Gap-ETH follows from ETH (or SETH), making it a strong conjecture. More discussion on this hypothesis and why it seems plausible to assume it (at least in the sense that countering it is currently far beyond our reach) can be found in *Appendix F* of [14].

1.1 The complexity landscape of exact computation

To properly state the main theorem that we build on, we need to define caterpillar graphs.

Figure 1.1: Two 4-caterpillar graphs: an out-caterpillar and an in-caterpillar.

Definition 3. *A λ-in-caterpillar is a digraph constructed as follows:*

- *take a path $(v_1, \ldots, v_\lambda)$ on λ vertices,*
- *add a new vertex w and connect it to any vertex v_i of the path by the arc wv_i. Repeat this step an arbitrary number of times.*

Similarly, we would define a λ-out-caterpillar, where the arcs formed in the second step lead out of the path vertices rather than into them. An out-caterpillar may be formed from an in-caterpillar by the reversal of edges. A 0-caterpillar is the empty graph.

[1] The algorithm's output is arbitrary if neither of the two cases occurs.

An illustration of this definition is given in Figure 1.1. Further, we define $\mathcal{C}_{\lambda,\delta}$ as the class of all directed graphs H such that after removing a subset $F \subseteq E_H$ of edges of size at most δ, $E_H \setminus F$ span a λ'-caterpillar for some $\lambda' \leq \lambda$.

The transitive closure of a directed graph arises by adding the uw edge whenever there are edges uv and vw for some vertices u, v, w (repeatedly, while applicable, so that the closure is another digraph). We call two directed graphs transitively equivalent if their transitive closures are isomorphic.

It is easy to see that a solution to the DSN for a specific pattern graph H simultaneously solves for all transitively equivalent graphs to H. Any choice of H from this equivalence class provides the same problem statement. Let $\mathcal{C}^*_{\lambda,\delta}$ denote the class of graphs transitively equivalent to elements of $\mathcal{C}_{\lambda,\delta}$. We can now state the result precisely:

Theorem 1. *([6, Theorem 1.2]) Let $\mathcal{H}$ be a recursively enumerable class of patterns (directed graphs).*

1. *If there are constants λ and δ such that $\mathcal{H} \subseteq \mathcal{C}^*_{\lambda,\delta}$, then $\mathcal{H}$-DSN can be solved optimally in time $2^{k+\tau\omega\log\omega}n^{\mathcal{O}(\omega)}$. Here, $k = |E_H|$ is the number of terminal pairs and we define $\omega = (1+\lambda)(1+\delta)$ and τ is the vertex cover number of the given input pattern $H \in \mathcal{H}$ (which is bounded by k).*
2. *Otherwise, if there are no such constants λ and δ, the problem is $W[1]$-hard for parameter k.*

1.2 Relevant classes of pattern graphs

Theorem 1 implicitly uses the observation that for $\mathcal{H}$-DSN, it makes no difference if we assume $\mathcal{H}$ to be closed under transitive equivalence. By similar reasoning we may require other conditions for $\mathcal{H}$. They are covered by the following:

Observation 1. *We may w.l.o.g. assume that the classes of pattern graphs we consider are closed under*

1. *transitive equivalence,*
2. *identifying terminals, meaning merging two or more into one, ignoring any edges between them,*
3. *taking path-preserving subgraphs (to be defined), and*
4. *reversing the direction of every edge.*

We will first discuss separately each point *2.* to *4.* After that, we prove that if a class $\mathcal{H}$ is *not* closed under these operations and $\mathcal{H}_c$ is the closure, there is a parameterized reduction from $\mathcal{H}_c$-DSN to $\mathcal{H}$-DSN, and it is therefore viable to assume we work with $\mathcal{H}_c$ in the first place: if we have an FPT algorithm for $\mathcal{H}$, we also do for $\mathcal{H}_c$. Naturally, the complexity changes (even the number of terminals itself changes by these operations), but for answering questions about FPT approximability, this doesn't matter. The reader may skip this technical proof, which we include for completeness and because Observation 1 will simplify our labour later.

To see that point *2.* is reasonable, consider a DSN instance (G, H, c) such that H may be obtained by identifying a certain set $\{u_1, \ldots, u_m\}$ of terminals of $H' \in \mathcal{H}$ into one terminal vertex v. We can equivalently solve this instance with

an instance (G', H', c'): G' arises from G by changing the vertex v (corresponding to the terminal $v \in H$) into a directed m-clique Q with zero-cost edges. Edges previously leading to and from v can lead into and from a fixed vertex in Q. In H', the identified terminals $\{u_1, \ldots, u_m\}$ shall correspond to the m vertices of Q. This solves (G, H, c), and any approximation factor guarantee for (G', H', c') translates directly to the same guarantee for this solution.

Point *3.*: we call a subgraph H of H' path-preserving if for each pair $u, v \in V_H \subseteq V'_H$, such that there is a path in H' from u to v, this whole path is also included in H.

The feasibility of this assumption follows similarly to *2.*: Let H be a path-preserving subgraph of H'. Assume a fixed homomorphism of H into H', so that $v \in H$ is also $v \in H'$. We may solve $\Delta = (G, H, c)$ with an instance (G', H', c'). H' is a supergraph of H, and so we only add into G' the vertices and edges of $H' \setminus H$. The added edges will have cost zero. As there is no path in H' between the vertices of its subgraph H, the instances are equivalent. Again, any approximation guarantee remains valid.

Point *4.*: if H arises from H' by reversing the direction of all edges, the instance $\Delta = (G, H, c)$ may be converted into $\Delta' = (G', H', c')$, such that also all edges in G' are reversed. Trivially, these instances are equivalent: reversing the edges of a solution to Δ' gives a solution of Δ with the same cost and vice versa.

Notice that the reductions discussed for these operations can be combined together, and so we may assume the closure not just for each separately, but for iterative application of any combination of these operations.

We now show an FPT reduction from $\mathcal{H}_c$-DSN to $\mathcal{H}$-DSN , where $\mathcal{H}_c$ is the closure of $\mathcal{H}$ under the operations of Observation 1. More precisely, we restrict ourselves to the assumption that our algorithms only deal with *decidable* classes of pattern graphs. This only seems natural, as it is unclear how an algorithm for $\mathcal{H}_c$-DSN would be built for undecidable $\mathcal{H}_c$. A looser restriction of $\mathcal{H}$ may be assumed however: for generality, we only require it to be recursively enumerable.

The statement of the following lemma and the proof is almost the same as that of [6, Lemma 5.2], only that they assume closure under a subset of the operations. We include the proof for completeness.

Lemma 2. *Let $\mathcal{H}$ be a recursively enumerable class of directed graphs. Let $\mathcal{H}_c$ be the closure of $\mathcal{H}$ under the operations of Observation 1 and let $\mathcal{H}'$ be a decidable subclass of $\mathcal{H}_c$. Then there is an FPT reduction from $\mathcal{H}'$-DSN to $\mathcal{H}$-DSN with parameter k.*

Proof. As a first remark, it is clear that detecting whether a graph H' with k' edges (and no isolated vertices) is equivalent, under iterative application of the operations of Observation 1, to a graph H on k edges, can be decided in time $h(k + k')$ for some h. Even a brute-force algorithm's running time depends only on the number of edges.

Fix an enumeration $H_1, H_2, \ldots$ of the digraphs of $\mathcal{H}$. Define $\varphi : \mathcal{H}_c \to \mathbb{N}$ as

$$\varphi(H') = \min\{i \in \mathbb{N} \mid H' \text{ can be obtained from } H_i \text{ using the four operations}\}.$$

Further, define $g : \mathcal{H}_c \to \mathbb{N}$ to be

$$g(H') = \max\{|V_{H_i}| \mid 1 \leq i \leq \varphi(H')\}.$$

Lastly, define

$$f(k) = \max\{\varphi(H') \cdot h(k + g(H')) \mid H' \in \mathcal{H}' \text{ such that } |V_{H'}| = k\}^2.$$

We remark that f only depends on k and the classes $\mathcal{H}, \mathcal{H}'$. It is also computable: as $\mathcal{H}'$ is decidable, we may iterate through all of its k-vertex patterns and enumerate $\mathcal{H}$ for each separately to determine $\varphi(H') \cdot h(k + g(H'))$.

The reduction itself proceeds as follows: given an instance $\Delta' = (G', H', c')$ of $\mathcal{H}'$-DSN (so that $H' \in \mathcal{H}'$), we enumerate $\mathcal{H}$ until finding a H convertible to H' under Observation 1. Observe that this requires at most $f(|V_{H'}|)$ steps.

Once we find such H, we also have the sequence of operations that convert H to H' and as discussed above, in the treatment of each operation, we have an instance $\Delta = (G, H, c)$ that is equivalent to Δ' in an approximation-preserving manner. □

The running-time of this reduction is huge of course. As stated, we mostly investigate whether $\mathcal{H}$-DSN does or does not have an FPT approximation algorithm for certain $\mathcal{H}$, with little emphasis on exact time complexity. We should not forget about this however, and neither about the fact that the operations of Observation 1 potentially change the parameter k.

On the other hand, we only use Lemma 2 for our approximation-hardness results, while our algorithms in Chapters 3 and 4 work with $\mathcal{H}$-DSN for classes $\mathcal{H}$ that will turn out to be naturally closed under these operations. The running time of this reduction doesn't therefore impact our results negatively.

1.3 Obstruction graphs

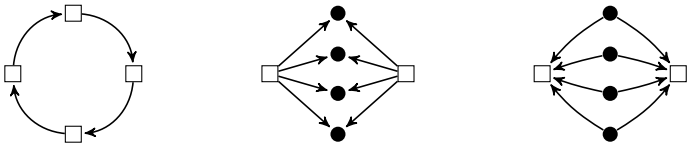

Figure 1.2: The "obstruction" graphs. A cycle and two diamonds, out-diamond and in-diamond, equivalent to each other by edge reversal.

To be able to show the inapproximability of $\mathcal{H}$-DSN for $\mathcal{H} \not\subseteq \mathcal{C}^*_{\lambda,\delta}$, we have to investigate the proof of case *2.* in Theorem 1. It is shown in the proof that if $\mathcal{H} \not\subseteq \mathcal{C}^*_{\lambda,\delta}$, its closure under operations from Observation 1 contains one of two "obstruction" graphs of arbitrary size. These graphs are defined as follows:

Definition 4. *(Obstruction graphs)*

- *A cycle as a pattern graph ensures that the vertices of any feasible solution are strongly connected. More precisely, for any two terminals t_1 and t_2, any feasible solution must contain a directed path from t_1 to t_2. This special case is the SCSS problem that we have already discussed.*

[2]It is relevant to ask how we define $f(k)$ when there is no graph in $\mathcal{H}'$ with k vertices. We will never use f in these cases however, so it may be defined arbitrarily there.

- *An out-diamond is formed as follows: take a set of m vertices $v_1, \ldots, v_m$ and two extra "root" vertices r_1 and r_2. For each v_i, add directed edges r_1v_i and r_2v_i. An in-diamond is formed from an out-diamond by edge reversal.*

These obstruction graphs are illustrated in Figure 1.2. Feldmann et al. [6] proved that if $\mathcal{H}$ is not a subset of $\mathcal{C}^*_{\lambda,\delta}$ for any fixed constants λ, δ, then the closure of $\mathcal{H}$ under the operations from Observation 1 includes every directed cycle or every diamond. More precisely, they only assume closure under transitive equivalence and identifying vertices. This yields five obstruction graph instead of two, but they are such that Observation 1 easily reduces them to the two stated.

These two types of instances are $W[1]$-hard for parameter k [6] (we will see stronger hardness results for both). The $W[1]$-hardness of DSN on classes that are not sub-classes of a $\mathcal{C}^*_{\lambda,\delta}$ follows: for each such class $\mathcal{H}$, we have a polynomial-time reduction (which is in fact identity) from one of the two $W[1]$-hard problems to $\mathcal{H}$-DSN.

1.4 Inapproximability of SCSS

In the light of the previous section, we may benefit from an inapproximability proof given by [7] for one of the two obstruction graphs: they show FPT-inapproximability for the Strongly Connected Steiner Subgraph problem.

Theorem 3. *([7, Theorem 1.9]) Assuming Gap-ETH, for any $\epsilon > 0$ and any computable function $f(k)$, there is no $f(k) \cdot n^{\mathcal{O}(1)}$-time algorithm that computes a $(2 - \epsilon)$-approximation for SCSS.*

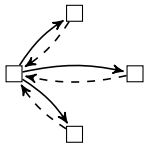

Figure 1.3: Using two star graphs to 2-approximate the SCSS problem. The in-star is drawn with dashed edges.

Notice that Theorem 1 immediately gives a matching approximation algorithm: if the given pattern graph is a cycle on the vertices $v_1, \ldots, v_m$ of a digraph G, it is equivalent to the following. Take an out-star leading from v_1 to the rest of vertices $v_2, \ldots, v_m$, and an in-star leading in the opposite direction. This is transitively equivalent to the cycle. The star graphs are 1-caterpillars, and so we may solve each separately, obtaining minimum-cost solutions. Since an SCSS satisfies the conditions given by each star, the optimum cost of each is at most the optimum cost of the solution to the SCSS. We thus obtain a 2-approximation algorithm. This approach is illustrated in Figure 1.3. It was already observed by [15]. We will reference this later.

Theorem 4. *The Strongly Connected Steiner Subgraph (SCSS) problem has an $\mathcal{O}^*(2^k)$-time 2-approximation algorithm.*

1.5 Other related work

To state other important results which we build on, we need to define the following.

Definition 5. Maximum Coloured Subgraph Isomorphism (MCSI)
Input: An instance $\Gamma = (G, \{V_i\}_{i=1}^{p}, H)$ *of MCSI consists of*

- *an undirected graph* $G = (V_G, E_G)$,
- *a partitioning* $V_1, \ldots, V_p$ *of the vertex set* V_G *into* p *"colours", and*
- *an undirected graph* $H = (V_H, E_H)$, *called the supergraph of* Γ, *with vertex set* $V_H = \{1, \ldots, p\}$.

Goal: For each colour class $i \in \{1, \ldots, p\}$, *find one vertex* $v_i \in V_i$ *in* G, *such that the number of edges* $ij \in E_H$ *of* H *(called superedges) for which* $v_i v_j \in E_G$, *is maximized.*

For a solution S, let $m(S) = \{v_i v_j : ij \in E_H\}$ be the set of edges in G satisfying the demands of H. Then we define the value of S to be $val(S) = \frac{|m(S)|}{|E_H|}$, the fraction of edges $ij \in E_H$ such that $v_i v_j$ is satisfied by S. The value of an instance $val(\Gamma)$ is the maximum value achievable by a solution to Γ, so that an instance with a solution satisfying all the demands given by H, has value 1.

The 2-ary Constraint Satisfaction Problem (2-CSP) may be seen as a reformulation of MCSI. Assuming the Gap-ETH, [16] established a hardness result for the 2−CSP, and thus also to MCSI. Building on the work of [7] and using a reduction from MCSI to DSN, [16] further give the following for the general DSN problem:

Theorem 5. *Assuming Gap-ETH, for any constant* $\rho > 0$, *and any function* f, *there is no* $f(k) \cdot n^{\mathcal{O}(1)}$*-time* $\frac{k^{1/4}}{2^{(\log k)^{1/2+\rho}}}$*-approximation algorithm for DSN.*

See *Corollary 4* in [16] for their discussion of this result: their proof yields this approximation lower bound for the special case where the pattern graph is the complete bipartite graph $H = K_{p,p} = (A, B, E)$ with edges oriented in the direction of the vertices of B. This way, $k = p^2$.

[17] discovered a way to solve DSN optimally by finding the shortest path in a cleverly constructed graph, obtaining an XP-time algorithm for this problem.

Theorem 6. *There is an* $\mathcal{O}(mn^{4k-2} + n^{4k-1} \log n)$*-time algorithm that finds the optimum for DSN, where* $n = |V_G|, m = |E_G|$.

Using a very different approach, [6] obtain an alternative XP-time algorithm for DSN.

There is also a polynomial-time algorithm developed by [18, Theorem 3.5], getting arbitrarily close to an $\mathcal{O}(\sqrt{k})$-approximation of DSN:

Theorem 7. *DSN can be approximated to within a factor of* $\mathcal{O}(k^{1/2+\epsilon})$ *in polynomial time, for any fixed* $\epsilon > 0$.

2. No approximation scheme

Theorem 1 provides two obstruction graphs, the SCSS and the diamond. It has been proven that the SCSS cannot be $(2-\epsilon)$-approximated in FPT time, if Gap-ETH holds. In this section, we prove a $\frac{4}{3}$-approximation lower bound for the second obstruction, the diamond, obtaining an inapproximability result for arbitrary $\mathcal{H}$-DSN, where $\mathcal{H}$ is not a subclass of $\mathcal{C}^*_{\lambda,\delta}$ for constants λ, δ.

We will need an inapproximability result known for the special case of MCSI (Definition 5) where H is a complete graph. It was shown by [7, Corollary 7.3].

Theorem 8. *Let $f, h : \mathbb{N} \to \mathbb{N}$ be any computable functions, such that $h(p) = o(1)$. Assuming randomized Gap-ETH, there is no $f(p) \cdot n^{\mathcal{O}(1)}$-time algorithm that, given an instance $\Gamma = (G, \{V_i\}_{i=1}^p, H)$ of MCSI where H is a p-clique, can distinguish between the cases*

- *(YES) $val(\Gamma) = 1$, and*
- *(NO) $val(\Gamma) < p^{-h(p)}$,*

2.1 Inapproximability of the diamond case

We denote the $\mathcal{H}$-DSN where $\mathcal{H}$ is the class of out-diamonds (the in-diamond case is equivalent by edge reversal) as "Diamond-DSN". The instance is formed by a graph G with non-negative edge costs, disjoint root vertices $r_1, r_2 \in V_G$ and a set of vertices $\mathcal{T} \subseteq V_G$, such that $r_1, r_2 \notin \mathcal{T}$. This specifies the pattern graph with $2|\mathcal{T}|$ edges, one from each root to each terminal vertex in $\mathcal{T}$.

In this section, we prove the following claim.

Theorem 9. *Assuming Gap-ETH, for any $\epsilon > 0$ and any computable function $f(k)$, there is no $f(k) \cdot n^{\mathcal{O}(1)}$-time algorithm that computes a $(\frac{4}{3} - \epsilon)$-approximation for Diamond-DSN.*

To achieve this, we combine the proof layout for Theorem 3 by [7] with a network construction of [6] (*in their proof of Lemma 5.12*).

Lemma 10. *For every constant $\gamma > 0$, there exists a polynomial time reduction that, given an instance $\Gamma = (G, H, V_1, \ldots, V_p)$ of MCSI where the subgraph H is a complete graph, produces an instance $(G', r'_1, r'_2, \mathcal{T}')$ of Diamond-DSN, such that*

- *(completeness) if $val(\Gamma) = 1$, then there exists a subgraph $N' \subseteq G'$ of cost $(3 + \gamma^{\frac{1}{5}})$ that is a feasible solution to the Diamond-DSN instance,*
- *(soundness) if $val(\Gamma) < \gamma$, then every network forming a feasible solution to our instance $(G', r'_1, r'_2, \mathcal{T}')$ of Diamond-DSN has cost more than $4(1-\gamma^{\frac{1}{5}})$, and*
- *(parameter dependency) the number of target terminals $|\mathcal{T}'|$ is $p(p-1)$. This makes the number of terminal pairs $2p(p-1)$.*

Proof. We assume w.l.o.g. that in G, there is no edge between any two vertices of the same partition V_j. We construct an instance of DSN where the pattern graph will be a diamond with two roots r'_1, r'_2 and the leaf (terminal) set $\mathcal{T}' = \{\ell_{ij} \mid 1 \le i, j \le p,\ i \ne j\}$. This way, we have $|\mathcal{T}'| = p(p-1)$, granting *parameter dependency*.

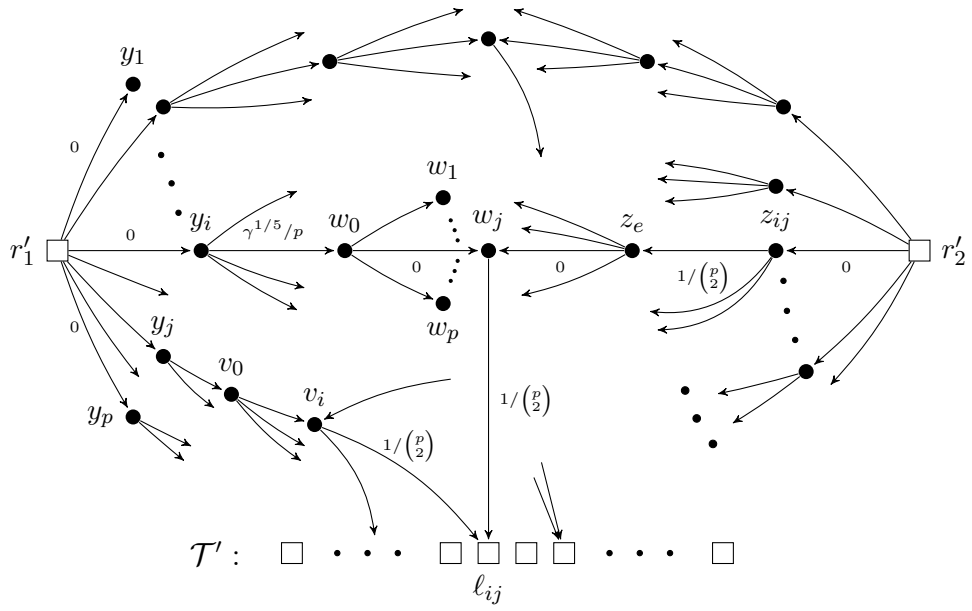

Figure 2.1: A schematic illustration of our reduction in Lemma 10. Only the edges of types $y_i w_0$, $z_{ij} z_e$ and $w_j \ell_{ij}$ are assigned nonzero costs.

Besides the terminal vertices, the constructed input graph G' contains the following.

- A vertex y_i for every $i \in \{1, \ldots, p\}$ representing the partition V_i. For each i, there is a zero-cost edge from r'_1 to y_i.
- For each $1 \leq i < j \leq p$, a vertex z_{ij}, to which r'_2 is connected with a zero-cost edge.
- Further, we put for each vertex w of some V_j a vertex "w_0" in G', and an edge of cost $\frac{\gamma^{1/5}}{p}$ connecting y_j to w_0 and
- For any vertex w of G, besides w_0, we put p vertices into G': w_1 up to w_p, and a zero-cost edge from w_0 to each of those p vertices.
- for each edge in G connecting a vertex of V_i to one of V_j, we put a vertex z_e, and an edge of cost $1/\binom{p}{2}$ from z_{ij} to z_e, in G'.
- For every $i, j \in \{1, \ldots, p\}$ such that $i \neq j$ and every $w \in V_i$, we have an edge $w_j \ell_{ij}$ of cost $1/\binom{p}{2}$, and
- for each vertex w, which occupies a certain V_i and each edge $e = wv$, where $v \in V_j$, we have a zero-cost edge $z_e w_j$ (thus, we also obtain the edge $z_e v_i$).

This construction is illustrated in Figure 2.1. Clearly, this DSN instance can be built in polynomial time. Except for our edge cost choices, it is exactly the same as the instance built by [6], and their *Figure 4* provides an alternative illustration of the situation. In our case, only three types of edges have positive costs.

(Completeness) If $val(\Gamma) = 1$, then there exists $(v_1, ..., v_p) \in V_1 \times ... \times V_p$ that induces a p-clique in Γ. A feasible solution to the constructed Diamond-DSN instance may, out of the positive-cost arcs, only include

- the p arcs of the form $y_i v_{i0}$ (for $i = 1, ..., p$),
- $\binom{p}{2}$ ones of the form $z_{ij} z_{\{v_i, v_j\}}$, corresponding to the $\binom{p}{2}$ edges of the clique, and analogously
- $2\binom{p}{2}$-cost arcs of the form $v_{ij} \ell_{ij}$ (and $v_{ji} \ell_{ji}$).

All this sums up to the claimed cost $\gamma^{1/5} + 1 + 2 = 3 + \gamma^{1/5}$.

(Soundness) Suppose that there is a network $N = (V(G'), E^*)$ of total cost $\varphi \leq 4(1 - \gamma^{\frac{1}{5}})$, solving our instance of Diamond-DSN.

Now let $\mathcal{S}_i \subseteq V_i$ be the set of all vertices $w \in V_i$ such that the arc $y_i w_0$ is included in E^* and let $\mathcal{S} := \mathcal{S}_1 \cup ... \cup \mathcal{S}_p$. Each arc of the form $y_i w_0$ has cost $\gamma^{1/5}/p$, so

$$|\mathcal{S}| \leq \frac{\varphi}{(\gamma^{1/5}/p)} < \frac{4}{(\gamma^{1/5}/p)} = 4\gamma^{-\frac{1}{5}} p.$$

If we consider every pair i, j such that $1 \leq i < j \leq p$, there need to be paths from r'_1 and r'_2 into ℓ_{ij} and ℓ_{ji}, as both are terminal. Only one type of arc (the $w_j \ell_{ij}$ type) leads into them, so for a given (i, j) pair, at least two such arcs (each of cost $1/\binom{p}{2}$) need to be included in the solution N. Another arc of this cost is necessary of the type $z_{ij} z_e$. By this simple reasoning, we see that even without considering the arcs of type $y_i w_0$, each pair (i, j) contributes to the total cost of N by at least $\frac{3}{\binom{p}{2}}$ units. Moreover, an arc of the $w_j \ell_{ij}$ type only leads to one terminal, and similarly, from z_e (connecting V_i and V_j), only the two terminals ℓ_{ij} and ℓ_{ji} are reachable. Thus, the positive-cost edges discussed for a pair (i, j) only contribute to connecting r'_1 and r'_2 to the two leaves ℓ_{ij} and ℓ_{ji}, i.e., the $3/\binom{p}{2}$ necessary units of cost are distinct for each pair.

As there are $\binom{p}{2}$ such pairs (i, j), the $1/\binom{p}{2}$-cost edges contribute to the cost of N, in total, by at least $\binom{p}{2} \cdot \frac{3}{\binom{p}{2}} = 3$ units. Since the cost of N is φ, at most $\binom{p}{2}(\varphi - 3)$ other edges of cost $1/\binom{p}{2}$ may be part of N.

Thus, there are at most $\binom{p}{2}(\varphi - 3) \leq \binom{p}{2}(1 - 4\gamma^{\frac{1}{5}})$ pairs (i, j) that require more than the three necessary $\frac{1}{\binom{p}{2}}$-cost arcs. To reformulate, at least $4\gamma^{\frac{1}{5}}\binom{p}{2}$ pairs require just 3 such arcs. We call the set of such pairs $\mathcal{P}$.

One more thing we need to observe is that for each $(i, j) \in \mathcal{P}$, there exists $u \in S_i$ and $v \in S_j$ such that $uv \in E_G$, but that is trivial, since the uv edge is exactly the one corresponding to the selected $z_{ij} z_e$-type arc in N. The u and v need to be part of S_i and S_j, since there is no other way r'_1 would be connected with ℓ_{ij} and ℓ_{ji}.

Now we just finish our analogy to the technical part of the proof of *Lemma 7.6* in [7]. Let $\phi : V_H \to V_G$ be a random assignment where each $\phi(i)$ is chosen independently uniformly at random from S_i. We saw that for each $(i, j) \in \mathcal{P}$, there exist vertices in S_i and S_j connected by an edge in G, meaning that for such a pair, the probability of $\phi(i)$ and $\phi(j)$ being connected is at least $\frac{1}{|S_i| \cdot |S_j|}$. We are going to use a special variant of Hölder's inequality for three variables[1], giving us

$$\begin{aligned}
(\sum_{(i,j)\in\mathcal{P}} \frac{1}{|S_i| \cdot |S_j|}) \cdot (\sum_{(i,j)\in\mathcal{P}} |S_i|) \cdot (\sum_{(i,j)\in\mathcal{P}} |S_j|) &\geq (\sum_{(i,j)\in\mathcal{P}} (\frac{1}{|S_i| \cdot |S_j|})^{\frac{1}{3}} \cdot |S_i|^{\frac{1}{3}} \cdot |S_j|^{\frac{1}{3}})^3 \\
&= (\sum_{(i,j)\in\mathcal{P}} 1^{\frac{1}{3}})^3 \\
&= |\mathcal{P}|^3.
\end{aligned}$$

[1] $(\sum_{r=1}^n a_r^3)(\sum_{r=1}^n b_r^3)(\sum_{r=1}^n c_r^3) \leq (\sum_{r=1}^n a_r b_r c_r)^3$

And we are able to lower-bound the expected number of superedges covered by ϕ as

$$\begin{aligned}\sum_{(i,j)\in\mathcal{P}} \frac{1}{|S_i|\cdot|S_j|} &\geq \frac{|\mathcal{P}|^3}{(\sum_{(i,j)\in\mathcal{P}}|S_i|)(\sum_{(i,j)\in\mathcal{P}}|S_j|)} \\ &\geq \frac{|\mathcal{P}|^3}{((p-1)|S|)^2} \\ &\text{(all } |S| \text{ total vertices for all } (p-1) \text{ } i\text{-s and } j\text{-s)} \\ &> \frac{(4\gamma^{\frac{1}{5}}\binom{p}{2})^3}{(p-1)^2\cdot(4\gamma^{-\frac{1}{5}}p)^2} \\ &\text{(by plugging in bounds for } |S| \text{ and } \mathcal{P}) \\ &= \gamma\binom{p}{2}.\end{aligned}$$

Hence, there exists a solution to Γ of density more than γ, proving that $val(\Gamma)$ is in fact above this bound. This is a contradiction.

All three desired properties are checked. □

Now we can prove Theorem 9 itself:
Suppose that, for some constant $\epsilon > 0$ and for some function $f(k)$ independent of n, there exists an $f(k)\ n^{\mathcal{O}(1)}$-time $(\frac{4}{3}-\epsilon)$-approximation algorithm $\mathbb{A}$ for Diamond-DSN.

Clearly, we can find $\gamma^* = \gamma^*(\epsilon)$ such that $\frac{4(1-\gamma^{*\frac{1}{5}})}{3+\gamma^{*\frac{1}{5}}} \geq (\frac{4}{3}-\epsilon)$.

Once more copying the approach of [7], we develop an algorithm $\mathbb{B}$ that contradicts *Theorem 8*, for $h(p) := \log(1/\gamma^*)/\log\ p$. Given an *MCSI* instance $(G, H, V_1, ..., V_p)$ (where H is a clique), $\mathbb{B}$ uses the reduction to Diamond-DSN of *Lemma 10* to obtain an instance on a digraph G' with $k = 2p(p-1)$ terminal pairs. Using the cost obtained with algorithm $\mathbb{A}$, our new algorithm $\mathbb{B}$

- answers YES if the cost is at most $4(1-\gamma^{*\frac{1}{5}})$
- answers NO if higher.

To see the correctness of $\mathbb{B}$, first observe that in the YES case ($val(\Gamma) = 1$), *Lemma 10* guarantees that the optimal solution in G' has cost at most $3+\gamma^{*\frac{1}{5}}$ and since $\mathbb{A}$ is a $(\frac{4}{3}-\epsilon)$-approximation algorithm, it returns a solution of cost at most $(3+\gamma^{*\frac{1}{5}}).(\frac{4}{3}-\epsilon) \leq 4(1-\gamma^{*\frac{1}{5}})$ by our choice of γ^*, i.e., $\mathbb{B}$ correctly answers YES.

In the NO case, our *soundness* property guarantees that the optimum in G' is more than $4(1-\gamma^{*\frac{1}{5}})$ and $\mathbb{B}$ thus also correctly answers NO.

This contradiction to *Theorem 8* concludes the proof of *Theorem 9*.

2.1.1 Conclusion

Unlike the SCSS case, we cannot show a matching approximation algorithm. The Diamond-DSN can trivially by 2-approximated again using two out-stars, whose optimum can be computed in FPT time. We are unaware of an approximation

algorithm guaranteeing a better factor. This hardness result only proves $(\frac{4}{3} - \epsilon)$-inapproximability however. It is unclear if the proof can be improved to get a tighter lower bound.

We have now established constant factor inapproximability for both obstruction graphs. This way we can strengthen the statement of Theorem 1:

Theorem 11. *Let $\mathcal{H}$ be a recursively enumerable class of patterns (directed graphs).*

1. *Either $\mathcal{H} \subseteq \mathcal{C}^*_{\lambda,\delta}$ for some constants λ, δ and we have an FPT algorithm for $\mathcal{H}$-DSN, or*
2. *there is no FPT approximation scheme for $\mathcal{H}$-DSN computing the $(1+\epsilon)$-approximation to this problem, assuming Gap-ETH.*

In the following chapters on approximation algorithms, we may therefore only wish for constant factor approximations at best.

3. Approximating paths

In this chapter, we 2-approximate $\mathcal{H}$-DSN for a special class $\mathcal{H}$ where the problem is W[1]-hard.

3.1 FPT 2-approximation for paths

We give an FPT 2-approximation algorithm for Path-DSN, where Path is the class of directed paths. We will use the following notation for this special case.

Definition 6. *An instance $\Pi = (G, H, c)$ of Path-DSN consists of*

- *a directed graph G with non-negative edge costs $c : E_G \to \mathbb{R}^+$ and*
- *a pattern graph H which is a directed path on vertices $v_1, \ldots, v_{k+1}$ of V_G.*

This special case of $\mathcal{H}$-DSN therefore uses the k demand pairs $(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_{k+1})$.

In this section we show the following.

Theorem 12. *There is a $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time 2-approximation algorithm for Path-DSN.*

This section may be seen as a prelude to the next one, which generalizes this result.

By identifying the first and last vertex of a path, we get a cycle. By forming a pattern graph as the cycle on all vertices, we can solve the general Traveling Salesman Problem, making Path-DSN not just APX-hard, but NPO-hard [19]. It is in fact easy to see that no polynomial-time constant-factor approximation exists for Path-DSN, unless P = NP: such factor would allow us to detect Hamiltonian paths in any graph. Theorem 1 shows W[1]-hardness of Path-DSN. Most notably, Theorem 3 says that 2 is the best approximation factor obtainable in FPT time (assuming Gap-ETH).

A strongly connected component (SCC) of a digraph is its inclusion-maximal strongly connected subgraph. Our algorithm relies on the following observation.

Lemma 13. *If an instance Π of Path-DSN is feasible (that is, it has at least one solution satisfying all constraints), it has an optimum solution N for which the following holds: the sequence $(v_1, \ldots, v_{k+1})$ may be partitioned into certain sub-sequences $V_1, V_2, \ldots, V_r$ respecting the original order (that is, if $v_i \in V_a$ and $v_j \in V_b$ for $a < b$, then $i < j$), such that*

- *for each $1 \leq a \leq r$, there is an SCC in N, which contains all vertices of V_a and no vertices of any other V_b for $b \neq a$. We will call this strongly connected component S_a.*
- *For each $1 \leq a < r$, N has a unique path P_a connecting S_a to S_{a+1}. This path contains exactly one vertex of S_a and exactly one vertex of S_{a+1}, and no vertices of any S_b for $b \notin \{a, a+1\}$.*

Knowing that we can 2-approximate the SCSS problem, our algorithm naturally follows from this lemma. It guesses the partition of the path vertices into

sub-sequences, forms 2-approximations of their SCCs and finds minimum-cost paths between them. All this can be done in the stated time.

We first prove the lemma and then give the precise description of the algorithm and the analysis of its correctness.

Proof. (Lemma 13)
We assume $\Pi = (G, (v_1, \ldots, v_{k+1}))$ is feasible, and therefore attains an optimum. Consider any optimum solution N. If an SCC of N contains vertices v_i and v_j where $i < j$, it also necessarily contains all the vertices $v_h,\ i < h < j$: this follows from the problem statement, since in N there must be a path from v_i to v_{i+1}, another from v_{i+1} to v_{i+2}, up to the path from v_{j-1} to v_j, but v_i and v_j being in the same strongly connected subgraph of N also implies a path from v_j to v_i, yielding a path from any of these vertices to any other. This means that any SCC inside N that contains some path vertex may only contain consecutive path vertices.

This observation partitions $(v_1, \ldots, v_{k+1})$ into sets $V_1, \ldots, V_r$ corresponding to SCCs $S_1, \ldots, S_r$ in N. The obtained sets may clearly be ordered in the desired way, as they only contain consecutive vertices of $(v_1, \ldots, v_{k+1})$.

If this produces more than just one sub-sequence $V_1 = \{v_1, \ldots, v_{k+1}\}$, then each terminal vertex in V_a $(a < r)$ must be connected to V_{a+1}, proving the existence of a directed path P in N from the SCC S_a to S_{a+1}. We may denote its sub-path containing only one vertex of S_a and one vertex of S_{a+1} as P_a. The path P_a cannot contain any vertex of another SCC S_b: such vertex would either show

- that V_{a+1} and V_b are parts of the same SCC in N, if $a + 1 < b$. Here, a sub-path of P goes from S_b into S_{a+1}, but the problem statement postulates that there is also some path in N going from V_{a+1} to V_b, showing an SCC including both S_{a+1} and S_b, in contradiction to the inclusion-maximality of both components.
- Or that V_b and V_a are contained in the same SCC if $b < a$, analogously.

Both cases yield a contradiction with our definition of the partition $V_1, \ldots, V_r$.

We see that the union of components S_a $(a \in \{1, \ldots, r\})$ and paths P_a (for $1 \leq a < r$) is feasible for Π, and therefore also optimal (N could additionally contain some zero-cost edges, if they exist in G). □

We proceed by describing the algorithm.

We guess the optimum partition of $V_G = V_1 \cup \ldots \cup V_r$ guaranteed by Lemma 13: a function $s : \{1, \ldots, k\} \to \{0, 1\}$ represents such a guess, where each corresponding part is induced by a maximal sequence of consecutive vertices $v_i, v_{i+1}, \ldots, v_{i+j}$ such that $s(\ell) = 0$ for each $i \leq \ell < i + j$. There are therefore a total of 2^k such guesses. Our algorithm iterates through all of them. Each iteration proceeds thus:

We have guessed the partition $V_1, \ldots, V_r$. We now apply dynamic programming. For an index $i \in \{1, \ldots, r-1\}$ and for an arbitrary vertex $u \in V_G$, we define the number $T(i, u) \in \mathbb{R}^+$ to be equal to the total cost of our 2-approximation of the minimum-cost subgraph of G where each V_j for $j \leq i$ is part of an SCC S_j, these SCCs are connected by directed paths P_j connecting S_j to S_{j+1} for $1 \leq j < i$, and the SCC containing V_i itself also contains the vertex u.

To compute $T(1, u)$ for $u \in V_G$, we use the 2-approximation (Theorem 4) for the minimum-cost SCSS containing $V_1 \cup \{u\}$. For $1 < i < r$, we get the recurrence

$$T(i, u) = \min_{v,w \in V_G} T(i-1, v) + SCC(V_i \cup \{u, w\}) + P(v, w),$$

where $SCC(S)$ is the total cost of our 2-approximate solution to SCSS for the set $S \subseteq V_G$ and $P(v, w)$ is the minimum-cost path between vertices v and w.

In other words, instead of finding the approximation for the SCC containing just the set V_i, we also "guess" the vertices v and w: the source and sink vertex of the path connecting SCCs S_{i-1} and S_i.

Once we have $T(r-1, u)$ for each $u \in V_G$, we get the 2-approximate cost for the guessed partition as

$$\min_{u,v \in V_G} T(r-1, u) + SCC(V_r \cup \{v\}) + P(u, v),$$

i.e., the guess of the source-sink vertices of the path connecting SCCs with V_{r-1} and V_r. Clearly, our SCSS approximations and optimal paths between them (from which we computed functions SCC, P and eventually T), can be reconstructed to obtain a corresponding feasible solution to the given Path-DSN instance.

It remains to prove that if we output the minimum-cost feasible solution over all guessed partitions $V_1, \ldots, V_r$, we obtain a 2-approximation.

Proof. (Theorem 12)
If the given instance Π was infeasible, we find no approximate solution and correctly answer that there is none. Otherwise, Lemma 13 guarantees an optimum solution with SCCs $S_1, \ldots, S_r$ and paths $P_1, \ldots, P_{r-1}$ connecting them. The cost of this solution is the sum of the costs of edges in each S_a and each P_a. These edge sets are disjoint, since

- an intersection in the edges of S_a and S_b $(a \neq b)$ contradicts the definition of an SCC,
- each P_a intersects S_a and S_{a+1}, each in one vertex, and therefore no edge, and no other $S_b, b \notin \{a, a+1\}$, and
- two P_a, P_b $(a < b)$ cannot share an edge: otherwise, P_b starts at a vertex of S_b and eventually reaches an edge of P_a, which leads to $S_{a+1} (a+1 \leq b)$. By the problem definition, there is a path from S_{a+1} to S_b. We see that there is a nontrivial path from S_b to itself. This contradicts the inclusion-maximality of the SCC S_b, as the path contains in addition at least one vertex of P_b.

One iteration guessed the corresponding partition $V_1, \ldots, V_r$. We prove that this iteration finds a 2-approximation of this optimum. Our output network has cost at most this 2-approximation.

For this guess, we proceed by iterating over $1 \leq i < r$ and computing $T(i, u)$ for each vertex u. We prove the approximation factor by induction on i: Assume P_1 starts at vertex $u \in S_1$ and ends at w. We found $T(1, u)$ by finding a 2-approximate SCSS containing $V_1 \cup \{u\}$. Since $S_1 \supseteq (V_1 \cup \{u\})$, $T(1, u)$ is less than or equal to double the cost of S_1.

Now assume this induction hypothesis: for $1 \leq i < r$ we found a $T(i, v)$ containing at most a 2-approximation of the sum of costs of S_j and P_k for $1 \leq j \leq i, 1 \leq k < i$, where v is the starting vertex of the path P_i. In the induction

step, we know we also once guess the end vertex w of P_i and the start vertex u of P_{i+1} (if $i < r$). For this guess, $T(i+1, u)$ is computed by finding the minimum-cost path between w and u, which has at most the cost of P_{i+1}, and the 2-approximate SCSS containing $V_{i+1} \cup \{u, w\}$, which has at most twice the cost of S_{i+1} (which contains $V_{i+1} \cup \{u, w\}$ by Lemma 13).

This concludes the induction step. The algorithm found a 2-approximate solution of Π for this guess of $V_1, \ldots, V_r$, and we output the solution with minimal cost over all guesses, which is at most the one for this guess. □

It is worth noting that to apply this reasoning, the guessing of the first and last vertex of each P_a is necessary. To see this, consider Figure 3.1:

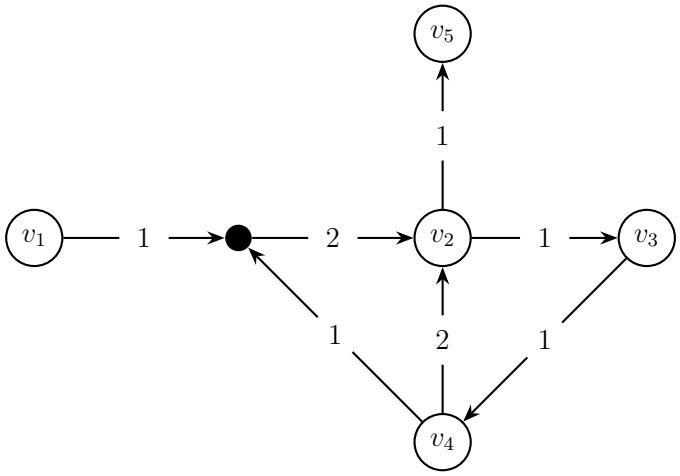

Figure 3.1: *A graph showing the necessity of guessing the source and sink vertices of paths connecting the SCCs. Costs are written for each edge. The black vertex is not included in the pattern graph.*

The optimum partition here is $V_1 = \{v_1\}, V_2 = \{v_2, v_3, v_4\}, V_3 = \{v_5\}$, but only finding the optimum SCC for V_2 would yield the triangle between its vertices, which is an SCC of cost 4. The total cost of thus obtained feasible solution is 8, while the real optimum is 7, which omits the edge connecting v_4 to v_2 and instead uses the SCC containing V_2 along with the black vertex - an SCC of total cost 5, the black vertex being the sink vertex of the used path between the obtained SCCs for V_1 and V_2.

Our proof relies on the fact that the algorithm finds a 2-approximation of each S_i of an optimum solution. Here we see that for certain graphs, minimum-cost SCSSes for just the partitions don't translate to minimum-cost solutions in total, making the reference to a 2-approximation of the optimum invalid without guessing the source and sink vertices of these paths.

The *condensation graph* C_H of a directed graph H is the DAG obtained by contracting each strongly connected component of H into a single vertex. Our algorithm for Path-DSN guesses which vertices belong to the same SCC. Therefore, if the pattern graph H wasn't a path, but only C_H was, the exact same algorithm would still be applicable (vertices of an SCC in H need to be in the same SCC in any feasible solution as well). With this observation, from now on we will work with condensations of pattern graphs, rather than the pattern graphs themselves.

3.2 2-approximation for multiple paths

We call an arc e of a directed graph H *redundant* if the graph $H - e$ is transitively equivalent to H. Observe that for a DAG D, the graph $\hat{D}$ that has no redundant arcs and is transitively equivalent to D, is unique: assuming there is another transitively equivalent $\hat{D}_2$, if we fix a topological ordering $u_1, \ldots, u_n$ of the n vertices of D, which is therefore a topological ordering of both $\hat{D}$ and $\hat{D}_2$, consider for contradiction an arc $u_i u_j$ w.l.o.g. in $\hat{D}_2$ that doesn't appear in $\hat{D}$. By transitive equivalence, there must be a path $(u_i, u_{\pi_1}, \ldots, u_{\pi_\ell}, u_j)$ in $\hat{D}$ (such that ℓ is at least 1 and for each q: $i < \pi_q < j$). Once more by transitive equivalence, there must be paths in $\hat{D}_2$ connecting u_i to u_{π_1}, u_{π_1} to u_{π_2} etc. up to one connecting u_{π_ℓ} to u_j. Since we have a valid topological order, $u_i u_j$ must be disjoint from these $\ell + 1$ paths. But since they connect u_i to u_j, $u_i u_j$ is a redundant arc, in contradiction to $\hat{D}$ not containing any redundant arcs.

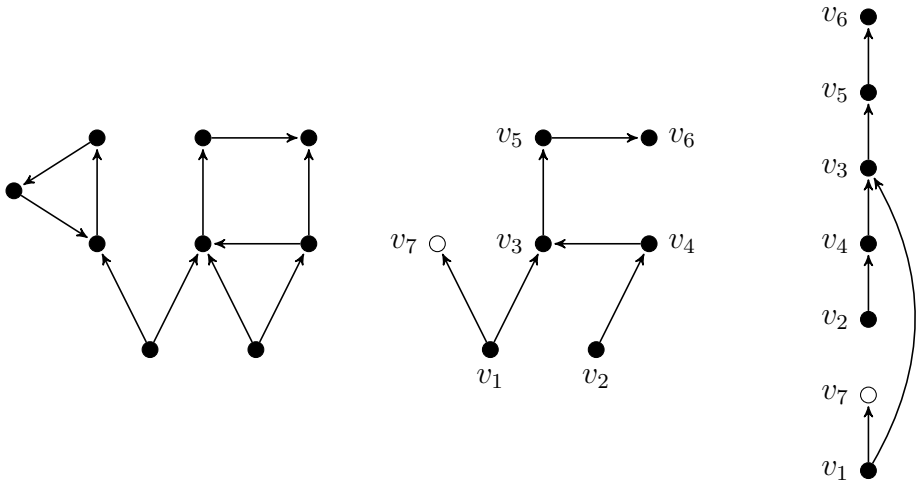

Figure 3.2: A possible pattern graph H, its condensation graph $\hat{C}_H$ with redundant edges removed, and an example topological ordering of $\hat{C}_H$. The vertex v_7 represents a contracted SCC. The pattern $\hat{C}_H$ is the union of three paths (two aren't sufficient).

Definition 7. *Given a directed graph H, we denote as $\hat{C}_H$ the graph obtained by removing redundant edges from the condensation graph C_H. We have seen that this defines a unique graph.*

p-Path is the class of directed graphs H such that the edges $E(\hat{C}_H)$ can be covered by at most p (possibly overlapping) paths. p-Path-DSN is then the DSN problem restricted to pattern graphs of this class.

Note that we do not require $\hat{C}_H$ to be the union of at most p paths, we only need its edges to be covered by the paths, and $\hat{C}_H$ may contain an arbitrary additional number of isolated vertices (corresponding to isolated SCCs in H).

Figure 3.2 gives an example of a pattern graph H in 3-Path. By combining our algorithm for Path-DSN with the XP-time algorithm of Theorem 6 for general DSN, we will obtain the following.

Theorem 14. *There is a $k! \cdot 4^k \cdot n^{\mathcal{O}(p)}$-time 2-approximation algorithm for p-Path-DSN, where k is the number of edges of the pattern graph H.*

The algorithm generalizes our approach to Path-DSN. We will again guess a partition of the terminal vertices. Previously, H was a path and gave a unique ordering of the guessed parts. Here, we will guess a topological ordering of the parts, to proceed analogously. Similarly to the Path-DSN's guessing of tail and head vertices of P_a in S_a and S_{a+1}, we will have to devise a guessing procedure for such vertices for each of the at most p paths. We will see in the correctness analysis how such guessing corresponds to certain minimal solutions to the problem instance Δ.

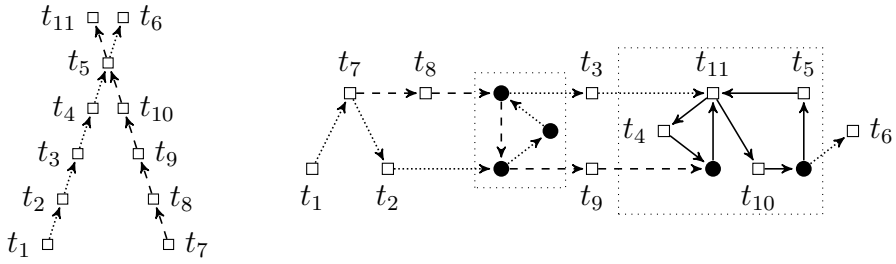

Figure 3.3: An example pattern graph H, formed by two paths, on the left. A possible minimal solution to it in some graph G on the right (edge costs omitted). Solid lines form an SCC containing terminals of both paths, which is enforced by the pattern. For the rest of edges, dashed ones are dedicated to one path and dotted ones to the other, both in the pattern and the solution.

Before describing the algorithm, we illustrate a simple pattern graph H and its possible solution in some G in Figure 3.3. The pattern graph H is the union of two paths of length six. The solution features two nontrivial strongly connected components (with more than one vertex), drawn inside dotted rectangles. The first is a byproduct of taking two paths between terminals, the second contains four terminal vertices and is enforced by the pattern graph.

The algorithm. On input, we get the instance $\Delta = (G, H, c)$, such that the edges of $\hat{C}_H$ are covered by at most p paths. We proceed as follows.

1. Iterate through all partitions $A = \{A_1, \ldots, A_r\}$ of the terminal vertex set V_H such that whenever $u, v \in A_i$ and there is in H a path from u to v through w, then also $w \in A_i$. In particular, vertices of the same SCC in H need to fall into the same part of A.
2. By contracting the vertices of each A_i into one and greedily removing redundant edges in the resulting graph, we obtain a modified pattern graph $\tilde{H}$ which is necessarily acyclic. Find $q \leq p$ paths $\tilde{P}_1, \ldots, \tilde{P}_q$ that cover all edges of $\tilde{H}$. This can always be done, as $\tilde{H}$ arises from $\hat{C}_H$ by contracting vertex sets, which can only shorten the paths present in $\hat{C}_H$.
3. Iterate through all valid topological orderings $t = (\tilde{u}_1, \ldots, \tilde{u}_r)$ of $V(\tilde{H})$. Assume that A is numbered in such a way that $\tilde{u}_i$ represents the set A_i.
4. With $\tilde{H}$ and its ordering $(\tilde{u}_1, \ldots, \tilde{u}_r)$ fixed, proceed by the following dynamic programming (DP) procedure.

The DP procedure introduces another level of guessing. For an index $1 \leq i \leq r$ in the ordering t and a q-tuple $J = (v_1, \ldots, v_q) \in V_G^q$ of vertices, the entry $D(i, J)$ of our DP table stores the cost of an approximate partial solution to Δ (corresponding to a certain prefix of each path of $\hat{C}_H$).

If $U \subset V(G)$ is a subset of vertices, let $S(U)$ denote the 2-approximate solution to SCSS given by Theorem 4, where the pattern graph is the cycle on all vertices of U. Furthermore, for $u_i, v_i \in V(G)$, let

$$P((u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m))$$

be the minimum-cost subgraph P of G that connects each u_i to each v_i. By Theorem 6, this can be computed optimally in $n^{\mathcal{O}(m)}$ time. In the case where $u_j = v_j$, the pair (u_j, v_j) doesn't result in any demand on P. Naturally, let $c(S(\ldots))$ and $c(P(\ldots))$ denote the cost of these subgraphs (that is, the sum of costs of their edges).

For each $1 \leq i \leq r$, let $B_i(J) = B_i(v_1, \ldots, v_q) := \{v_b \mid \tilde{u}_i \text{ lies on } \tilde{P}_b\}$. This is how we compute the entries of D:

- The base case: Let $F(1, J) := S(A_1 \cup B_1(J))$ and store the value $D(1, J) = c(F(1, J))$
- For $i > 1$, there will be further guessing. We need to iterate through all ordered subsets $C \subseteq (J \setminus B_i(J))$. Each index in J corresponds to one of the q paths. We may imagine $C \cup B_i(J)$ as $(v_{\sigma(1)}, \ldots, v_{\sigma(\ell)})$. Given an ℓ-tuple of vertices $(v'_{\sigma(1)}, \ldots, v'_{\sigma(\ell)}) \in V(G)^\ell$, denote for each $1 \leq i \leq q$

$$w_j := \begin{cases} v'_j & \text{if } v_j \in B_i(J) \ \cup \ C, \\ v_j & \text{otherwise.} \end{cases}$$

We compute the entry $D(i, J)$ as

$$D(i, J) = \min_{\substack{J'=(u_1,\ldots,u_q) \\ \in V_G^q}} \ \min_{C \subseteq (J \setminus B_i(J))} \ \min_{\substack{\text{choices of } v'_j \\ \forall j:\ v_j \in B_i(J) \cup C}} c(S(A_i \cup B_i(J) \cup C \cup \bigcup_{j: v_j \in C} w_j)) + c(P((u_1, w_1), \ldots, (u_q, w_q))).$$

The minimum is obtained for some $J' \in V_G^q$, and let $S = S(\ldots)$ and $P = P(\ldots)$ for the minimum term. We then define $F(i, J) = F(i-1, J') \cup S \cup P$.

Notice that each $F(i, J)$ satisfies all demands of $\tilde{H}$ that appear in t until the i-th index. This way, $F(r, J)$ is a feasible solution to Δ for each J, given that Δ is a feasible instance (otherwise, we detect infeasibility). The algorithm outputs the minimum-cost $F(r, J)$ obtained for any iteration.

For summary, the whole procedure is sketched in Algorithm 1.

Running time. In the DP routine, we iterate for each i through the n^q choices of J, then there are another n^q choices of J' on level $i-1$. We need at most $2^q \leq n^q$ iterations to enumerate the choices of C, and the set $B_i(J) \cup C$ has at most q elements, so we have to iterate through at most n^q choices of the v'_i-s. We then use the $2^k \cdot n^{\mathcal{O}(1)}$ algorithm of Theorem 4 to compute S, and the $n^{\mathcal{O}(q)}$ algorithm to compute P. Summing up, the DP routine's running time is upper-bounded by $r \cdot n^{4q} \cdot 2^k n^{\mathcal{O}(1)} \cdot n^{\mathcal{O}(q)} = 2^k \cdot n^{\mathcal{O}(p)}$.

A brute-force way to iterate through all choices in steps 1. and 3. is to try all (at most) $k!$ permutation of the vertices of $\hat{C}_H$ and for each that is a valid topological ordering, partition it into sub-sequences to obtain the partition A. This is technically done in the opposite order compared to the algorithm, but for the sake of computing the running time, this counts every valid partition of A, as the requirement of $w \in A_i$ whenever there is a $u \to w \to v$ path for $u, v \in A_i$

guarantees a topological ordering of $\hat{C}_H$ where all sets A_i occupy consecutive indices.

This multiplies to the total running time upper bound $k! \cdot 2^k \cdot 2^k \cdot n^{\mathcal{O}(p)} = k!\ 4^k\ n^{\mathcal{O}(p)}$, as promised.[1]

We note that if $\hat{C}_H$ was the union of at most p paths (with no additional isolated vertices), the number of valid topological orderings may be upper-bounded by p^k rather than $k!$. The most important fact for us however is that the algorithm runs in FPT time, for greater generality (and to allow isolated vertices in $\hat{C}_H$), we adopt the running time $k!\ 4^k\ n^{\mathcal{O}(p)}$.

Correctness. Assume the feasibility of the instance $\Delta = (G, H, c)$. Consider its minimal solution $N \subseteq G$ (such as any edge-minimal optimum solution) and its condensation graph C_N.

- Fix a topological ordering $t_N = (u_1, \ldots, u_m)$ of C_N.
- Each node $u_i \in V(C_N)$ represents a set $U_i \subseteq V_N \subseteq V_G$, in the SCC S_i.
- For each i, let $V_i \subseteq U_i$ be the *terminal* vertices contained in U_i (a V_i may therefore be empty).
- Construct a graph $\hat{H}$ from H by contracting the vertices of each V_i into one $\hat{u}_i$ and then removing all redundant edges. For an empty $V_j = \emptyset$, add an isolated vertex $\hat{u}_j$ into $\hat{H}$. This is so that $(\hat{u}_1, \ldots, \hat{u}_m)$ is a valid topological ordering of $\hat{H}$.
- Because $H \in p$-Path, the edges of $\hat{H}$ may be covered by $q \leq p$ paths $\hat{P}_1, \ldots \hat{P}_q$. Any such $\hat{P}_j$ is the concatenation of edges of the form $\hat{u}_a\hat{u}_b$, which induce a demand on any feasible solution (and N in particular). For each such edge $\hat{u}_a\hat{u}_b$, we may fix in N a path $Q_{a,b}$ from S_a to S_b. The concatenation of such paths with arbitrary paths in each strongly connected component S_b, connecting the last vertex of $Q_{a,b}$ to the first of $Q_{b,c}$, results in a path $P_j \subseteq N$ (induced by $\hat{P}_j$).
- By edge-minimality, N is the union of paths P_1 up to P_q and SCCs S_1 up to S_m. Observe, that if $V_i = \emptyset$, all edges of S_i are already covered by the paths $P_1, \ldots, P_q$, by edge-minimality of N.
- We now introduce the notion of *layers* of N. Let $(a_j)_{j=1}^r$ be the increasing sequence of exactly the indices $1 \leq i \leq m$ such that $V_i \neq \emptyset$. That is, S_{a_j} is the j-th strongly connected component in t_N that contains terminal vertices.

  For each $1 \leq i \leq r$, consider each path P_b $(1 \leq q \leq p)$ and the position of its vertices in the order t_N. If the first vertex v of P_b appears only *after* u_{a_i} in t_N, define $v_{i,b} = v$. Otherwise, let $v_{i,b}$ be the last vertex of P_b in the order t_N to appear at index at most a_i. We slightly abuse the notation, as some vertices of P_b appear in some S_j and aren't directly represented in t_N. We understand the statement of *appearing last* in t_N as being the last vertex of t_N represented by u_ℓ, which is either u_{a_i} itself, or such that the index $\ell \leq a_i$ is maximal.

  For $1 < i \leq r$, if $v_{i,b} \in U_{a_i}$, define $w_{i,b}$ to be the *first* vertex of P_b in U_{a_i}. Otherwise, let $w_{i,b} := v_{i,b}$.

[1] In this analysis, we didn't compute the time needed to cover $E(\hat{C}_H)$ by $q \leq, p$ paths. Covering the edges of a DAG by a minimum number of paths can be done in polynomial time using topological induction. The algorithm isn't completely trivial, but we skip its description, because even a brute-force approach only adds an $f(k)$ factor, keeping the running-time FPT.

We illustrate the notions of $v_{i,b}$ and $w_{i,b}$ in figure 3.4: we imagine a certain topological ordering t_N of the strongly connected components of N. Seven of the SCCs, $S_{a_1}, \ldots, S_{a_7}$, contain terminal vertices, drawn as rectangles. Intuitively said, $v_{i,b}$ is the last vertex of P_b that appears in t_N at most as far as S_{a_i} (or we define it as the first vertex of P_b, if the path only starts after S_{a_i} in t_N). We label a vertex $w_{i,b}$ in the Figure if P_b passes through S_{a_i} and $w_{i,b}$ is its first vertex in S_{a_i}. With this definition, a vertex may have several such labels.

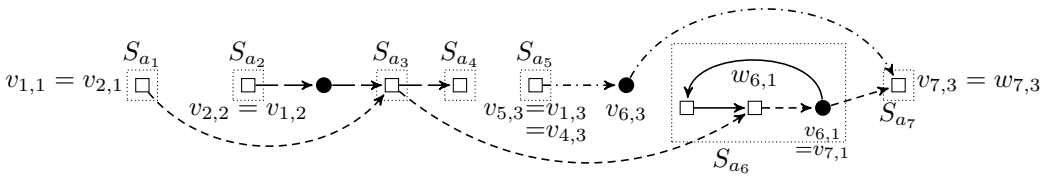

Figure 3.4: We illustrate the notion of $v_{i,b}$ and $w_{i,b}$ on a topological ordering of a possible solution N. Here, N is the union of three paths. The paths indexed as P_1 and P_2 are drawn using two types of dashed lines. Path P_3 is dash-dotted. There are also two edges drawn using solid lines. They serve to satisfy the same demands as P_1, but in fixing each P_i, we only take simple paths.

- The first layer L_1 is defined as S_1 (notice that a_1 must be equal to 1). For $2 \leq i \leq r$, we define L_i as the union of S_{a_i} and, for every path P_b, its subpath $P_{i,b}$ connecting $v_{i,b}$ to $w_{i,b}$. Each L_i is thus a subgraph of N.

We observe that for $i < j$, the layers L_i and L_j are edge-disjoint: First, note that no $P_{\ell,b}$ shares an edge with any SCC S_{a_j} (that is, an SCC in F that contains a positive number of terminals), since we defined these subpaths in such a way that only their first and last vertex may lie in an S_{a_j}. Further, two SCCs S_{a_j} and S_{a_ℓ} are clearly edge-disjoint (even vertex-disjoint), otherwise they form one strongly connected subgraph of G (and neither one is thus a strongly connected *component* by itself). Now consider two paths, $P_{\ell,a}$ and $P_{j,b}$, such that $\ell < j$ (but possibly $a = b$). Assume they share a common edge e. In particular $P_{j,b}$ starts in a vertex $v_{j-1,b}$ of $S_{a_{j-1}}$, and eventually reaches e. Since $e \in P_{\ell,a}$, we may further follow this subpath and reach the vertex $w_{\ell,a}$ of S_{a_ℓ}. We have $\ell \leq j-1$, and two cases may occur:

- either $\ell < j-1$, but then we have just observed a path from U_{j-1} to U_ℓ in F, making T an invalid topological ordering of C_F,
- or $\ell = j-1$, in which case there is a cycle from $S_{a_{j-1}}$ into itself, which contains e. In particular, e is part of the SCC $S_{a_{j-1}}$. But we have already proven that $P_{\ell,a}$ must be edge-disjoint from $S_{a_{j-1}}$, which e now contradicts.

We reach a contradiction in both cases, so L_i is indeed edge-disjoint from L_j. Consider the iteration of our algorithm where in A, each A_j is equal to some V_ℓ (we consider this *the right guess* of A, corresponding to N). With this choice, our algorithm produces $\tilde{H}$ by contracting the vertices of each A_j, and in iterating through the topological orderings t of $(V(\tilde{H}))$, one t follows the order of t_N.[2] For $1 \leq i \leq r$, let $J_i = (v_{i,1}, \ldots, v_{i,q})$. Using induction, we will prove that

[2]Each $\tilde{u}_j$ corresponds to some $N(\tilde{u}_j)$ in N. We require t to be such that $\tilde{u}_a <_t \tilde{u}_b$ iff $N(\tilde{u}_a) <_{t_N} N(\tilde{u}_b)$.

with this ordering t, the computed $D(i, J_i)$, which is the cost of an approximate subsolution, will be at most twice the cost of $N_i = \bigcup_{j=1}^{i} L_j$ (for each i).

For $i = 1$, this is trivial, or rather a corollary of Theorem 4: We know that $N_1 = S_1$ must contain the vertices of V_i and of $B_i(J_i)$. By Theorem 4, a 2-approximation for such SCSS is computed as one of the candidates for $D(1, J_i)$, and we take the minimum-cost candidate.

The case $i > 1$ is also not difficult: by the induction hypothesis,

$$D(i-1, J_{i-1}) \leq 2 \cdot c(F_{i-1}).$$

Our DP routine considers all choices of J'. Fix $J' = J_{i-1} = (u_{i-1,1}, \ldots, u_{i-1,q})$ in particular. All vertices of $B_i(J_i)$ necessarily appear in S_{a_i}, so for one choice of C, the set $B_i(J_i) \cup C$ is exactly the subset of J_i contained in S_{a_i}, and the algorithm further considers the choices $v'_j = w_{i,j}$ for $v_{i,j} \in B_i J \cup C$.

We compute the 2-approximate SCSS

$$S = S(A_i \cup B_i(J_i) \cup C \cup \bigcup_{j : v_{i,j} \in C} w_{i,j})$$

and the minimum-cost solution

$$P = P((u_{i-1,1}, w_{i,1}), \ldots, (u_{i-1,q}, w_{i,q})).$$

The layer L_i is edge-disjoint from N_{i-1} and consists of the SCC S_{a_i} and $P_N := \bigcup_{j=1}^{q} P_{i,j}$. We defined S_{a_i} and P_N in such a way that they are edge-disjoint. We thus see that

$$\begin{aligned} c(N_i) &= c(N_{i-1}) + c(S_N) + c(P_N), \text{ and therefore} \\ D(i, J_i) &\leq D(i-1, J_{i-1}) + S + P \leq 2 \cdot c(N_{i-1}) + 2 \cdot c(S_N) + c(P_N) \leq 2 \cdot c(N_i). \end{aligned}$$

The graph $F \subseteq G$ that we output is the minimum-cost feasible solution we find, and as such has cost at most $D(r, J_r) \leq 2 \cdot c(N)$. It is therefore a 2-approximation.

Algorithm 1 Outline of the algorithm of Theorem 14

Require: DSN instance $\Delta = (G, H, c)$, such that $E(\hat{C}_H)$ is covered by $q \leq p$ paths.

1: **for each** valid partition $A = \{A_1, \ldots, A_r\}$ of V_H **do**
2: Construct $\tilde{H}$
3: ▷ *(from H by contracting vertices of each A_i into one, and removing redundant edges greedily)*
4: **for each** valid topological ordering $t = (\tilde{u}_1, \ldots, \tilde{u}_r)$ of $V(\tilde{H})$ **do**
5: Run DP routine to obtain F.
6: ▷ *(that is, compute $D(i, J)$ for each $1 \leq i \leq r$ and $J \in V_G^q$)*
7: **end for**
8: **end for**
9: Output the minimum-cost F found (if Δ is feasible).

4. Unbounded path count

Our 2-approximation of p-Path-DSN covers a significant class of pattern graphs. It is by far not general, however. In particular, an out-star (that is, a 1-out-caterpillar), for which we have an exact FPT algorithm by Theorem 1, may require an arbitrary number of paths. We extend the notion of caterpillar graphs to an unbounded length.

4.1 Long caterpillars

Only the condensation of a pattern graph will matter to us.

Definition 8. *We call a digraph H an out-caterpillar if $\hat{C}_H$, its condensation graph with redundant edges removed, is a λ-out-caterpillar for arbitrary λ.*

We define in-caterpillars analogously.

Any in-caterpillar can be obtained from an out-caterpillar by edge reversal. Therefore, let Cat denote w.l.o.g. the class of out-caterpillars. We will design a 3-approximation algorithm for Cat-DSN.

The algorithm. We first convert the pattern graph H to $\hat{C}_H$, which is a λ-caterpillar. This caterpillar is the union of a path on λ vertices, and the other vertices, the caterpillar's "legs". We denote the set of path vertices as $P \subseteq V(\hat{C}_H)$, and the set of leg vertices will be L. For any vertex u of $\hat{C}_H$, let $H(u)$ denote the vertex set of the SCC of H that u represents and let for $U \subseteq \hat{C}_H$, $H(U)$ be the union of sets $H(u)$ for $u \in U$.

We follow the following guessing procedure.

1. Iterate through all such partitions $\{V_1, V_2, \ldots, V_m\}$ of $V(\hat{C}_H)$, that whenever $u, v \in V_i$ and there is in $\hat{C}_H$ a path from u to v through w, then also $w \in V_i$. Each V_i represents the vertices $H(V_i)$.
2. For one such partition, mark its parts as

$$\{V_1, \ldots, V_m\} = \{P_1, \ldots, P_{m_1}, L_1, \ldots, L_{m_2}\},$$

   such that $P_1, \ldots, P_{m_1}$ are precisely the parts in which there is at least one vertex of P. Furthermore, let this numbering of the P_i follow the order in which the path vertices appear in the path (if a P_i contains vertices $u, v \in P$, it also contains all vertices on the path that appear between u and v, so this is a valid requirement). The sets $L_1, \ldots L_{m_2}$ then only contain leg vertices of the caterpillar. We present an example of such partition in Figure 4.1: a caterpillar's vertices are partitioned into eight parts, drawn inside dotted areas.
3. Use the algorithm of Theorem 14 to find a 2-approximate solution F' for the union of the path $(P_1, \ldots, P_{m_1})$ and the guessed strongly connected components $L_1, \ldots L_{m_2}$.[1]

[1] The non-redundant condensation $\hat{C}_{H'}$ of the pattern graph H' given to this sub-procedure is the union of a path and isolated vertices, so H' is a 1-path.

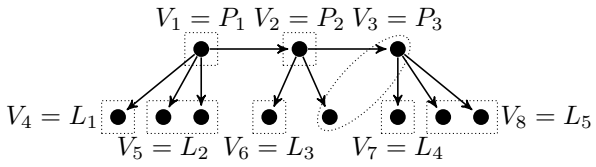

Figure 4.1: A 3-caterpillar with an example partition of vertices into eight parts $V_1, \dots V_8$. Each part is also labeled as P_i or L_j depending on whether it contains path vertices.

4. The pattern $\hat{C}_H$ is an out-caterpillar, so for each $v \in L$, there is a uv edge in $\hat{C}_H$ for some $u \in P$. For every L_i, define

$$P(L_i) = \max\{j \mid \exists uv \in E(\hat{C}_H) \text{ s.t. } v \in L_i \text{ and } u \in P_j\}$$

as the farthest SCC on the path P that is demanded to be connected to L_i by the pattern graph H.
Iterate through all m_2-tuples $(\alpha_1, \dots, \alpha_{m_2})$, such that for each i,

$$\alpha_i \in [P(L_i), m_1].$$

5. Modify G into G' by setting the costs of all edges featured in F' to 0. For each P_i, fix a vertex $p_i \in P_i$, similarly for each L_j, fix $\ell_j \in L_j$.
6. For each $1 \leq i \leq m_1$, let $S_i := \{\ell_j \mid \alpha_j = i\}$.
Separately for each $1 \leq i \leq m_1$, use the algorithm of Theorem 1 to find the minimum-cost subgraph F'_i of G' satisfying the DSN demands given by an out-star (i.e., 1-out-caterpillar) pattern graph formed by the $|S_i|$ demand pairs $p_i\ell_j$ for each $\ell_j \in S_i$. Only the cost function c changed for G', so $F'_i \subseteq G$.
The subgraph $G \supseteq F := F' \cup \bigcup_i F'_i$ now satisfies all demands of H and serves as a candidate for an approximate solution.
7. Once all iterations are processed, output the minimum-cost F found in any iteration.

The algorithm thus relies on two levels of guessing: we guess the partition of $V(\hat{C}_H)$ into parts forming SCCs in F, and we guess the order $(\alpha_1, \dots, \alpha_{m_2})$ of leg vertices as they appear in F.

Correctness. Assume feasibility of the given instance $\Delta = (G, H, c)$. Consider its minimal solution $N \subseteq G$ (such as any edge-minimal optimum solution). For a node $u \in \hat{C}_H$, all terminals of $H(u)$ need to be strongly connected in N. Therefore, $V(\hat{C}_H)$ may be partitioned into subsets $\{V_1, \dots, V_m\}$ such that for each i, $H(V_i)$ is the set of all terminals appearing in one SCC of N. By the DSN problem's definition, if $u, v \in V_i$ and there is a path from u to v via w in $\hat{C}_H$, then also $w \in V_i$. Furthermore, The V_i-s that feature vertices of the path P may be renamed and ordered as they appear on P: $P_1, \dots, P_{m_1}$. The other SCCs (with only leg vertices) may be renamed to $L_1, \dots L_{m_2}$ ($m_1 + m_2 = m$). This is precisely how we viewed our guess when describing the algorithm.

Theorem 14 guarantees that the partial solution F' we obtain in step 3. is a 2-approximation for the pattern formed from the path $(P_1, \dots, P_{m1})$ and isolated SCCs L_1 up to L_{m_2}. The solution N features both the path and the SCCs, and so the cost $c(F')$ is at most twice the cost $c(N)$.

By the problem statement, to each (SCC containing) L_j there leads a path in N from some (SCC containing) P_i. This i need be at least $P(L_j)$, using our former notation, as H has an edge uv for $u \in H(P_i)$ and $v \in H(L_i)$. For each j, let C_j denote the path in N connecting P_i to L_j (with exactly one vertex p_j of P_i and one ℓ_j of L_j in C_j) for i maximal, and let α_j denote this maximal i. Denote as N_i the union

$$N_i = \bigcup_{j:\ \alpha_j = i} C_j.$$

We may observe that for two indices $p < q$, the vertex-intersection $N_p \cap N_q$ is empty: Consider a vertex v in both a path C_p of N_p and C_q of N_q. But we defined C_p to be the path from the furthest P_i on the path P to some SCC L_j. The intersection in v shows that there is a path from P_q into L_j, contradicting that C_p belongs to C_p.[2]

Consider the iteration of the algorithm (step 4.) where $(\alpha_1, \ldots, \alpha_{m_2})$ is exactly as denoted in the last paragraph. Our partial solution F' connects all vertices of each P_i and each L_j into SCCs, so that adding to F' a path from an arbitrary $p_i \in P_i$ to an arbitrary $\ell_j \in L_j$ (as performed by the algorithm) amounts to connecting any other vertex of P_i to any one of L_j. Theorem 1 then says that the cost $c(F'_i)$ is at most $c(N_i)$. And since the N_i-s are vertex-disjoint (and therefore edge-disjoint), the cost of the union $c(\bigcup_i F'_i)$ is at most $\sum_i c(N_i) = c(\bigcup_i N_i) \leq c(N)$.

Summing all terms up, we have

$$c(F) = c(F' \cup \bigcup_i F'_i) \leq c(F') + c(\bigcup_i F'_i) \leq 2 \cdot c(N) + c(N) = 3 \cdot c(N).$$

Each F found by the algorithm (in any final iteration) is a feasible solution of Δ. We output the minimum-cost one found, and so in particular one with cost at most $3 \cdot c(N) = 3 \cdot OPT$, considering an edge-minimal optimum solution N.

Running time. As we have proven in the previous chapter, the number of partitions produced in step 1. of our algorithm is at most $2^k \cdot k!$. Using Theorem 14 in step 3. to compute a 2-approximate 1-Path-DSN sub-procedure requires time $4^k \cdot k! \cdot n^{\mathcal{O}(1)}$. Ignoring polynomial steps, we further have to guess the α_i-s in step 4. As an upper-bound, there are at most $k!$ choices of such m_2-tuples. For each such guess, we have to compute the (at most k) out-stars (1-out-caterpillars) F'_i. This results in a further factor of $k \cdot 2^k \cdot n^{\mathcal{O}(1)}$. Summing up, the total time is $2^k \cdot k!(4^k \cdot k! \cdot n^{\mathcal{O}(1)} + k \cdot k! \cdot 2^k \cdot n^{\mathcal{O}(1)})$, and we can conclude:

Theorem 15. *There is an $8^k \cdot (k!)^2 \cdot n^{\mathcal{O}(1)}$-time 3-approximation algorithm for the Cat-DSN problem, where k is the number of demand pairs.*

4.1.1 Multiple caterpillars

Following the previous chapter, it is natural to extend our result to pattern graphs formed by several caterpillars. Define p-Cat to be the class of pattern graphs H such that $\hat{C}_H$ is the union of at most p caterpillars (we may allow both *in-* and

[2]Furthermore, this intersection hints at a redundancy in C_p (making it intersect C_q), contradicting the edge-minimality of N.

out-caterpillars). Any instance of p-Cat-DSN may be approximated by finding the $q \leq p$ caterpillars covering $\hat{C}_H$ and running the algorithm of Theorem 15 q times, resulting in the following:

Corollary 1. *There is an $f(k) \cdot n^{\mathcal{O}(1)}$-time $3p$-approximation algorithm for the p-Cat-DSN problem, where k is the number of demand pairs.*

Theorem 5 gives us a lower-bound for the approximability of p-Cat-DSN: As discussed, a more elaborate statement of Theorem 5 deals with the class $\mathcal{K}$ of complete bipartite graphs $H = K_{p,p} = (A, B, E)$ with edges oriented in the direction of the part B. The theorem states that for any constant $\rho > 0$, there is no $f(p) \cdot n^{\mathcal{O}(1)}$-time $\frac{p^{1/2}}{2^{(2 \log p)^{1/2+\rho}}}$-approximation algorithm for $\mathcal{K}$-DSN (if Gap-ETH holds).

Since each directed complete bipartite graph $K_{p,p} \in \mathcal{K}$ is the union of p out-stars,[3] it is also in p-Cat. We obtain the following.

Corollary 2. *Assuming Gap-ETH, for any function f and any constant $\rho > 0$, there is no $f(p)$ $n^{\mathcal{O}(1)}$-time $\frac{p^{1/2}}{2^{(2 \log p)^{1/2+\rho}}}$-approximation algorithm for p-Cat-DSN.*

We obtained this lower bound in a very trivial way: the $K_{p,p}$ is the union of simple out-stars. It doesn't use the potential of p arbitrarily long caterpillars.

We should remark that in the same trivial way, we can get a matching lower bound, under the less established *Strongish Planted Clique Hypothesis (SPCH).* [20] proved the following:

Theorem 16. *([20, Corollary 14.]) Assuming SPCH, there is no $f(p) \cdot n^{\mathcal{O}(1)}$-time algorithm that can approximate $\mathcal{K}$-DSN to within a factor of $o(p)$ for any function f, where $\mathcal{K}$ is the class of directed complete graphs $K_{p,p}$, as above.*

This gives the immediate corollary.

Corollary 3. *Assuming SPCH, there is no $f(p) \cdot n^{\mathcal{O}(1)}$-time algorithm that can approximate p-Cat-DSN to within a factor of $o(p)$ for any function f.*

This would imply that Corollary 1 gives an asymptotically optimal approximation ratio for p-Cat-DSN.

Using XP time, however, we can improve the approximation bound. We need another definition.

Definition 9. *Let (χ, π)-Cat denote the class of patterns (directed graphs) H, such that the edges of $\hat{C}_H$ can be covered by the union of at most κ caterpillars[4] and at most π paths.*

Now recall how we 3-approximated Cat-DSN in Theorem 15. We separately found a 2-approximation for the path in the caterpillar and the out-stars leading from this path into the *leg* components $L_1, \ldots L_{m_2}$. We may slightly modify this algorithm as follows.

[3] One for each vertex of the part A, with p edges branching to all vertices of B. Equivalently, it's the union of p in-stars, one for each vertex of B.

[4] Each either an in- or out-caterpillar of arbitrary length.

1. Given instance $\Delta = (G, H, c)$, iterate through all such partitions $\{V_1, V_2, \ldots, V_m\}$ of $V(\hat{C}_H)$, that whenever $u, v \in V_i$ and there is in $\hat{C}_H$ a path from u to v through w, then also $w \in V_i$.
2. Contract the vertices of each V_i into one, thus obtaining the graph $\tilde{H}$. Decompose $\tilde{H}$ into $c \leq \chi$ caterpillars, $p \leq \pi$ paths and possibly d isolated components $I_1, \ldots I_d$ (we may have an arbitrary number of them). More precisely, the i-th of the c caterpillars is formed by the path $P_i^{\textsc{Cat}}$, leg components $L_{i,1}$ up to L_{i,ℓ_i} and edges leading from $P_i^{\textsc{Cat}}$ into the leg components (as discussed in the previous section). Name the j-th of the p paths $P_j^{\textsc{Path}}$.
3. Instead of computing approximations for the caterpillars one by one, find the 2-approximate solution F' for all paths and isolated SCCs at once. That is, use Theorem 14 as a sub-procedure for the pattern graph given by the union of all $P_i^{\textsc{Cat}}$-s, all $P_j^{\textsc{Path}}$-s, $L_{i,m}$-s, and all I_q-s. This pattern is in $(c+p)$-Path, and so this can be done in time $f(k) \cdot n^{\mathcal{O}(c+p)}$.
4. Modify G into G' by setting all weights of edges in F' to zero.
5. One by one, for each $1 \leq i \leq c$, proceed exactly as in Theorem 15: Imagine we are only approximating the i-th caterpillar. The path $P_i^{\textsc{Cat}}$ has already been computed and we have to connect the legs of the caterpillar by isolated out-stars (or in-stars). The subgraph F_i' obtained this way (as the union of DSN solutions for separate star pattern graphs) has cost at most OPT.
6. Output the union of F' and $\bigcup_{i=1}^{c} F_i'$.

This leads to the following.

Theorem 17. *There is an $f(k) \cdot n^{\mathcal{O}(\chi+\pi)}$-time $(2+\chi)$-approximation algorithm for (χ, π)-Cat-DSN.*

The argumentation is exactly the same as in Theorem 15, and we will not repeat it precisely.

4.2 Special cases

The (χ, π)-Cat-DSN problem and its approximation by Theorem 17 generalizes all of our previous approximation algorithms. (χ, π)-Cat is a superclass of the class $\mathcal{C}^*_{\lambda,\delta}$ for which DSN is FPT-solvable by Theorem 1, and it also contains the directed complete bipartite graphs $K_{p,p}$ for which Theorems 5 and 16 give strong approximation lower bounds.

What lies between the two? It is a challenging open problem to give the full approximation complexity landscape for $\mathcal{H}$-DSN. We now propose two particular classes of pattern graphs that seem to be significant. We believe that resolving the FPT approximability of these special cases would give important insight into the general problem.

4.2.1 d-Diamonds

Definition 10. *A d-out-diamond is a directed graph formed as follows: Take a set $v_1, \ldots, v_m$ of m vertices and d other "root" vertices $r_1, \ldots, r_d$. The edge set will be exactly*

$$\{r_i v_j \mid 1 \leq i \leq r,\ 1 \leq j \leq m\}.$$

Let δ-DIAMOND be the class of d-out-diamonds for $d \leq \delta$.

See Figure 4.2 for an illustration. The focus on out-diamonds is without loss of generality, as reduction from in-diamonds is trivial by edge-reversal.

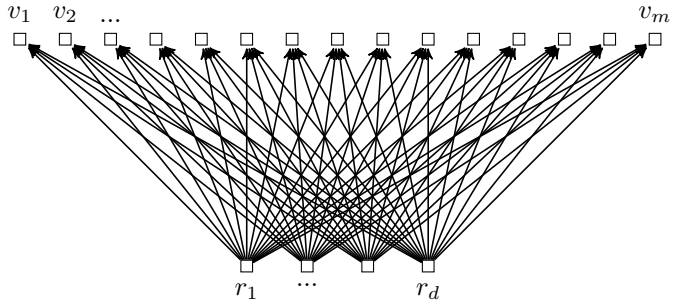

Figure 4.2: A d-out-diamond is the union of d out-stars.

The 1-DIAMOND-DSN is the well studied DIRECTED STEINER TREE (DST), which is NP-hard by a trivial reduction from the undirected STEINER TREE (of Karp's seminal paper [4]). Polynomial-time polylogarithmic inapproximability of DST is shown by [21]. On the other hand, DST is FPT by 1.

We dedicated Section 2.1 to showing FPT $(\frac{4}{3} - \epsilon)$-inapproximability of 2-DIAMOND-DSN under Gap-ETH. This problem already yields open questions: we do not have a matching upper bound. We can trivially 2-approximate by two out-stars (this way, we can d-approximate any d-diamond), but we do not know whether this ratio can be improved, or how to find a tighter inapproximability bound.

A special case of a δ-DIAMOND is the complete directed bipartite graph $K_{\delta,\delta}$, which is $f(\delta) \cdot n^{\mathcal{O}(1)}$-time inapproximable to within polynomial factors in δ by Theorems 5 and 16.

Exposition to this problem, including the observations we just stated, suggests it being likely that hardness of approximation increases with δ. We propose the following conjecture.

Conjecture 2. *There is a constant $\alpha > 0$ such that for any function f, there is no $f(k) \cdot n^{\mathcal{O}(1)}$-time $(\alpha \cdot \delta)$-approximation algorithm for δ-DIAMOND-DSN, where k is the number of demands.*

In fact, we can't see how XP time in δ helps substantially decrease the approximation ratio of this problem, and we strengthen our conjecture further:

Conjecture 3. *There is a constant $\beta > 0$ such that for any functions f, g, there is no $f(k) \cdot n^{g(\delta)}$-time $(\beta \cdot \delta)$-approximation algorithm for δ-DIAMOND-DSN, where k is the number of demands.*

Not having tight approximability bounds for δ-DIAMOND-DSN poses a large gap in our knowledge of $\mathcal{H}$-DSN in general. Proving or contradicting these conjectures is thus important for approaching the full approximation complexity landscape of this general problem.

4.2.2 Grids

One other interesting subclass of p-Cat is the following.

Definition 11. *For any $m \geq 1$, a p-grid of length m is a directed graph formed thus: Take p vertex-disjoint directed paths $P_1, \ldots, P_p$ of length m, where $P_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,m})$. Add edges $v_{i,j}v_{i+1,j}$ for every $1 \leq i < p$ and $1 \leq j \leq m$. See Figure 4.3 for an illustration. Notice that this is a DAG with no redundant edges.*

Let π-Grid be the class of p-grids for $p \leq \pi$ (and arbitrary length m).

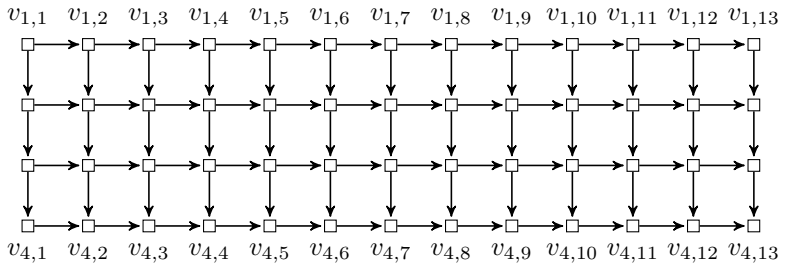

Figure 4.3: Example of a 4-grid of length $m = 13$.

For a $\pi \geq 2$, π-Grid is not a subclass of p-Path for any constant p. That is to say that with increasing length m, we get an arbitrary number of paths necessary to cover the edges of the grid.

We can cover a p-grid by $p - 1$ caterpillars and an additional path, where the caterpillars have the special property that each path vertex is connected to only one leg vertex. On the other hand, the connections in the grid allow for far less generality of the π-Grid-DSN compared to its superclass $(\pi - 1, 1)$-Cat-DSN.

Theorem 1 gives an FPT 3π-approximation algorithm for π-Grid-DSN (and Theorem 17 gives $(1 + \pi)$-approximation using time XP in π). We do not know if this approximation bound can be improved substantially (in FPT time, or in XP time $f(k) \cdot n^{g(\pi)}$). Neither is it clear how to obtain a strong approximation lower bound for this special case.

Conclusion

We have elaborated on several previous results. In Chapter 2, we extended Theorem 1 into the following.

For a recursively enumerable class $\mathcal{H}$ of directed graphs,

1. *Either $\mathcal{H} \subseteq \mathcal{C}^*_{\lambda,\delta}$ for some constants λ, δ and there is an FPT algorithm for $\mathcal{H}$-DSN, parameterized by the number of demands k, computing its optimum solution,*
2. *or there is no FPT $(\frac{4}{3} - \epsilon)$-approximation algorithm for $\mathcal{H}$-DSN, assuming Gap-ETH. In particular, this implies that there is no FPT $(1 + \epsilon)$-approximation scheme for this problem.*

In Section 3.2, we extended Theorem 6:

There is a $k! \cdot 8^k \cdot n^{\mathcal{O}(p)}$-time 2-approximation algorithm for pattern graphs formed by p paths (and other more general graphs).

Then in Section 4.1.1, we extrapolated into several long caterpillars, rather than paths, obtaining an asymptotically optimal approximation ratio under SPCH.

This is still far from a complete *approximation complexity landscape* which would provide the approximation equivalent of Theorem 1. In Section 4.2, we suggest directions that may lead beyond the results of this thesis.

Bibliography

[1] Hervé Kerivin and Ali Ridha Mahjoub. Design of survivable networks: A survey. *Networks*, 46(1):1–21, 2005.

[2] Wen-Tsuen Chen and Nen-Fu Huang. The strongly connecting problem on multihop packet radio networks. *IEEE Trans. Commun.*, 37(3):293–295, 1989.

[3] Ram Ramanathan and Regina Hain. Topology control of multihop wireless networks using transmit power adjustment. In *Proceedings IEEE INFOCOM 2000, The Conference on Computer Communications, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Reaching the Promised Land of Communications, Tel Aviv, Israel, March 26-30, 2000*, pages 404–413. IEEE Computer Society, 2000.

[4] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

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

[6] Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed steiner network problems. *CoRR*, abs/1707.06808, 2017.

[7] Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected steiner network problems. *ACM Trans. Algorithms*, 17(2):12:1–12:68, 2021.

[8] Esther Galby, Sándor Kisfaludi-Bak, Dániel Marx, and Roohani Sharma. Subexponential parameterized directed steiner network problems on planar graphs: a complete classification. *CoRR*, abs/2208.06015, 2022.

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

[10] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[11] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[12] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[13] Eli Ben-Sasson and Madhu Sudan. Short pcps with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

[14] Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 743–754. IEEE Computer Society, 2017.

[15] Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. *CoRR*, abs/1308.3520, 2013.

[16] Irit Dinur and Pasin Manurangsi. Eth-hardness of approximating 2-CSPs and directed steiner network. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[17] Jon Feldman and Matthias Ruhl. The directed steiner network problem is tractable for a constant number of terminals. *SIAM J. Comput.*, 36(2):543–561, 2006.

[18] Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. *ACM Trans. Algorithms*, 7(2):18:1–18:17, 2011.

[19] Pekka Orponen and Heikki Mannila. On approximation preserving reductions: Complete problems and robust measures (revised version). 1987.

[20] Pasin Manurangsi, Aviad Rubinstein, and Tselil Schramm. The strongish planted clique hypothesis and its consequences. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 10:1–10:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[21] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 585–594. ACM, 2003.

List of Abbreviations

- DAG: Directed acyclic graph
- DP: Dynamic programming
- DSN: The Directed Steiner Network problem (see Definition 1)
- ETH: The Exponential-Time Hypothesis
- FPT: Fixed-Parameter Tractable
- Gap-ETH: A strengthening of the ETH (see Conjecture 1)
- SCC: Strongly connected component (an inclusion-maximal strongly connected subgraph of a directed graph)
- SCSS: Strongly Connected Steiner Subgraph (see Definition 4)
- SPCH: Strongish Planted Clique Hypothesis
- XP: Slicewise Polynomial Time