Many Visits TSP Revisited

Łukasz Kowalik
Institute of Informatics, University of Warsaw, Poland
https://www.mimuw.edu.pl/~kowalik/
kowalik@mimuw.edu.pl

Shaohua Li
Institute of Informatics, University of Warsaw, Poland
shaohua.li@mimuw.edu.pl

Wojciech Nadara
Institute of Informatics, University of Warsaw, Poland
w.nadara@mimuw.edu.pl

Marcin Smulewicz
Institute of Informatics, University of Warsaw, Poland
m.smulewicz@mimuw.edu.pl

Magnus Wahlström
Royal Holloway, University of London, UK
Magnus.Wahlstrom@rhul.ac.uk

Abstract

We study the Many Visits TSP problem, where given a number $k(v)$ for each of n cities and pairwise (possibly asymmetric) integer distances, one has to find an optimal tour that visits each city v exactly $k(v)$ times. The currently fastest algorithm is due to Berger, Kozma, Mnich and Vincze [SODA 2019, TALG 2020] and runs in time and space $\mathcal{O}^*(5^n)$. They also show a polynomial space algorithm running in time $\mathcal{O}(16^{n+o(n)})$. In this work, we show three main results:

- A randomized polynomial space algorithm in time $\mathcal{O}^*(2^n D)$, where D is the maximum distance between two cities. By using standard methods, this results in a $(1+\epsilon)$-approximation in time $\mathcal{O}^*(2^n \epsilon^{-1})$. Improving the constant 2 in these results would be a major breakthrough, as it would result in improving the $\mathcal{O}^*(2^n)$-time algorithm for Directed Hamiltonian Cycle, which is a 50 years old open problem.
- A tight analysis of Berger et al.'s exponential space algorithm, resulting in an $\mathcal{O}^*(4^n)$ running time bound.
- A new polynomial space algorithm, running in time $\mathcal{O}(7.88^n)$.

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

Keywords and phrases many visits traveling salesman problem, exponential algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.66

Funding This workshop was supported by a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 714704 (PI: Marcin Pilipczuk).
Łukasz Kowalik: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).
Shaohua Li: Supported by ERC Starting Grant CUTACOMBS (Grant Agreement No 714704).
Wojciech Nadara: Supported by ERC Starting Grant CUTACOMBS (Grant Agreement No 714704).
Marcin Smulewicz: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).

Acknowledgements The research leading to the results presented in this paper was partially carried out during the Parameterized Algorithms Retreat of the University of Warsaw, PARUW 2020, held in Krynica-Zdrój in February 2020.

1 Introduction

In the Many Visits TSP (MVTSP) we are given a set V of n vertices, with pairwise distances (or costs) $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$. We are also given a function $k : V \to \mathbb{Z}_+$. A valid tour of length ℓ is a sequence of vertices $(x_1, \ldots, x_\ell)$, where $\ell = \sum_{v \in V} k(v)$, such that each $v \in V$ appears in the sequence exactly $k(v)$ times. The cost of the tour is $\sum_{i=1}^{\ell-1} d(x_i, x_{i+1}) + d(x_\ell, x_1)$. Our goal is to find a valid tour with minimum cost.

Many Visits TSP is a natural generalization of the classical (asymmetric) Traveling Salesman Problem (TSP), which corresponds to the case when $k(v) = 1$ for every vertex v. Similarly as its special case, MVTSP arises with a variety of applications, including scheduling [24, 17, 6, 27, 12], computational geometry [20] and parameterized complexity [21].

1.1 Related work

The standard dynamic programming for TSP of Bellman [1], Held and Karp [16] running in time $\mathcal{O}^*(2^n)$ can be easily generalized to MVTSP resulting in an algorithm with the running time of $\mathcal{O}^*(\prod_{v \in V}(k(v)+1))$, as noted by Psaraftis [24]. A breakthrough came in the work of Cosmadakis and Papadimitriou [7] who presented an algorithm running in time $2^{O(n \log n)} + \mathcal{O}(n^3 \log \ell)$ and space $2^{O(n \log n)}$, thus essentially removing the dependence on the function k from the bound (the $\log \ell$ factor can be actually skipped if we support the original algorithm with a today's state-of-the-art minimum cost flow algorithm). This may be surprising since the *length* of the output sequence is ℓ. However, beginning from the work of Cosmadakis and Papadimitriou we consider MVTSP with compressed output, namely the output is a multiplicity function which encodes the number of times every edge is visited by the tour. By using a standard Eulerian tour algorithm we can compute an explicit tour from this output.

The crux of the approach of Cosmadakis and Papadimitriou [7] was an observation that every solution can be decomposed to a minimal connected spanning Eulerian subgraph (which enforces connectivity of the solution) and a subgraph satisfying appropriate degree constraints (which completes the tour so that the numbers of visits agree). Moreover, once we guess the degree sequence δ of the Eulerian subgraph, our task splits into two separate tasks: finding a cheapest minimal connected Eulerian subgraph consistent with δ (which is computationally hard) and finding a cheapest subgraph satisfying the degree constraints (which can by solved in polynomial time by a reduction to minimum cost flow).

Yet another breakthrough came only recently, namely Berger, Kozma, Mnich and Vincze [3, 2] improved the running time to $\mathcal{O}^*(5^n)$. Their main contribution is an idea that it is more convenient to use outbranchings (i.e. spanning trees oriented out of the root) to force connectivity of the solution. The result of Berger et al. is the first algorithm for MVTSP which is optimal assuming Exponential Time Hypothesis (ETH) [18], i.e., there is no algorithm in time $2^{o(n)}$, unless ETH fails. Moreover, by applying the divide and conquer approach of Gurevich and Shelah [15] they design a polynomial space algorithm, running in time $\mathcal{O}(16^{n+o(n)})$.

1.2 Our results

In this work, we take the next step in exploration of the Many Visits TSP problem: we aim at algorithms which are optimal at a more fine grained level, namely with running times of the form $\mathcal{O}(c^n)$, such that an improvement to $\mathcal{O}((c-\epsilon)^n)$ for any $\epsilon > 0$ meets a kind of natural barrier, for example contradicts the Strong Exponential Time Hypothesis (SETH) [19] or the Set Cover Conjecture (SCC) [8]. Our main result is the following theorem.

▶ **Theorem 1.1.** *There is a randomized algorithm that solves* Many Visits TSP *in time* $\mathcal{O}^*(2^n D)$ *and polynomial space, where* $D = \max\{d(u,v) : u,v \in V, d(u,v) \neq \infty\}$. *The algorithm returns a minimum weight solution with constant probability.*

The natural barrier in this case is connected with Directed Hamiltonicity, the problem of determining if a directed graph contains a Hamiltonian cycle. Indeed, this is a special case of Many Visits TSP with $D = 1$, so an improvement to $\mathcal{O}^*(1.99^n D)$ in Theorem 1.1 would result in an algorithm in time $\mathcal{O}^*(1.99^n)$ for Directed Hamiltonicity. While it is not known whether such an algorithm contradicts SETH or SCC, the question about its existence is a major open problem which in the last 58 years has seen some progress only for special graph classes, like bipartite graphs [4, 9].

At the technical level, Theorem 1.1 uses the so-called algebraic approach and relies on two key insights. The first one is to enforce connectivity not by guessing a spanning connected subgraph as in the previous works, but by applying the Cut and Count approach of Cygan et al [10]. The second insight is to satisfy the degree constraints using the Tutte matrix [26, 22].

By using standard rounding techniques, we are able to make the algorithm from Theorem 1.1 somewhat useful even if the maximum distance D is large. Namely, we prove the following.

▶ **Theorem 1.2.** *For any* $\epsilon > 0$ *there is a randomized* $(1+\epsilon)$*-approximation algorithm that solves* Many Visits TSP *in* $\mathcal{O}^*(2^n \epsilon^{-1})$ *time and polynomial space.*

In Theorems 1.1 and 1.2 the better exponential dependence in the running time was achieved at the cost of sacrificing an $\mathcal{O}(D)$ factor in the running time, or the optimality of the solution. What if we do not want to sacrifice anything? While we are not able to get a $\mathcal{O}^*(2^n)$ algorithm yet, we are able to report a progress compared to the algorithm of Berger et al. in time $\mathcal{O}^*(5^n)$. In fact we do not show a new algorithm but we provide a refined analysis of the previous one. The new analysis is tight (up to a polynomial factor).

▶ **Theorem 1.3.** *There is an algorithm that solves* Many Visits TSP *in time and space* $\mathcal{O}^*(4^n)$.

In short, Berger et al.'s polyspace $\mathcal{O}^*(16^{n+o(n)})$ time algorithm iterates through all $O(4^n)$ degree sequences of an outbranching, finds the cheapest outbranching for each sequence in time $O(4^{n+o(n)})$, and completes it to satisfy the degree constraints using a polynomial time flow computation. Note that it is hard to speed up the cheapest outbranching routine, because for the sequence of $n-1$ ones and one zero we get essentially the TSP, for which the best known polynomial space algorithm takes time $O(4^{n+o(n)})$ [15]. However, we are still able to get a significant speed up of their algorithm, roughly, by using a more powerful minimum cost flow network, which allows for computing the cheapest outbranchings in smaller subgraphs.

▶ **Theorem 1.4.** *There is an algorithm that solves* Many Visits TSP *in time* $\mathcal{O}^*(7.88^n)$ *and polynomial space.*

Organization of the paper. In Section 3 we show that, essentially, using a polynomial time preprocessing step we can reduce an instance of Many Visits TSP to an equivalent one but with demands k bounded by $O(n^2)$. This reduction is a crucial prerequisite for Section 4 where we prove Theorem 1.1. Next, in Section 5 we prove Theorem 1.3 and in Section 6 we prove Theorem 1.4. We note that in these two sections we do not need the reduction from Section 3, however, in practice, applying it should speed-up the flow computations used in both algorithms described there. Finally, in Section 7 we show Theorem 1.2 and we discuss further research in Section 8.

2 Preliminaries

We use Iverson bracket, i.e., if α is a logical proposition, then the expression $[\alpha]$ evaluates to 1 when α is true and 0 otherwise.

For two integer-valued functions f, g on the same domain D, we write $f \leq g$ when $f(x) \leq g(x)$ for every $x \in D$. Similarly, $f + g$ (resp. $f - g$) denote the pointwise sum (difference) of f and g. This generalizes to functions on different domains D_f, D_g by extending the functions to $D_f \cup D_g$ so that the values outside the original domain are 0.

For a cost function $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$, and a multiplicity function $m : V^2 \to \mathbb{Z}_{\geq 0}$ we denote the cost of m as $d(m) = \sum_{u,v \in V^2} d(u, v)m(u, v)$.

Multisets. Recall that a *multiset* A can be identified by its *multiplicity* function $m_A : U \to \mathbb{Z}_{\geq 0}$, where U is a set. We write $e \in A$ when $e \in U$ and $m_A(e) > 0$. Consider two multisets A and B. We write $A \subseteq B$ when for every $e \in A$ we have $e \in B$ and $m_A(e) \leq m_B(e)$. Also, $A = B$ when $A \subseteq B$ and $B \subseteq A$. Assume w.l.o.g. that m_A and m_B have the same domain U. Operations on multisets are defined by the corresponding multiplicities as follows: for every $e \in U$, we have $m_{A \cup B}(e) = \max\{m_A(e), m_B(e)\}$, $m_{A \cap B}(e) = \min\{m_A(e), m_B(e)\}$, $m_{A \setminus B}(e) = \max\{m_A(e) - m_B(e), 0\}$, $m_{A \triangle B}(e) = m_{(A \setminus B) \cup (B \setminus A)} = |m_A(e) - m_B(e)|$. This notation extends to the situation when A or B is a set, by using the indicator function $m_A(e) = [e \in A]$.

Directed graphs. Directed graphs (also called digraphs) in this paper can have multiple edges and multiple loops, so sets $E(G)$ will in fact be multisets. We call a directed graph *simple* if it has no multiple edges or loops. We call it *weakly simple* if it has no multiple edges or multiple loops (but single loops are allowed). For a digraph G by $G^{\downarrow}$ we denote the *support* of G, i.e., the weakly simple digraph on the vertex set $V(G)$ such that $E(G^{\downarrow}) = \{(u, v) \mid G \text{ has an edge from } u \text{ to } v\}$.

Given a digraph $G = (V, E)$ we define its *multiplicity function* $m_G : V^2 \to \mathbb{Z}_{\geq 0}$ as the multiplicity function of its edge multiset, i.e., for any pair $u, v \in V$, we put $m_G(u, v) = m_E((u, v))$. Conversely, for a function $m : V^2 \to \mathbb{Z}_{\geq 0}$ we define the *thick graph* $G_m = (V, E)$ so that $m_G = m$. Abusing notation slightly, we will identify m and G_m, e.g., we can say that m is strongly connected, contains a subgraph, etc.

We call a directed graph *connected* if the underlying undirected graph is connected. Similarly, a *connected component* of a digraph G is a subgraph of G induced by a vertex set of a connected component of the underlying undirected graph.

For a graph G (directed or undirected) and a subset $X \subseteq V(G)$, by $G[X]$ we denote the subgraph induced by X.

Solutions. The following observation follows easily from known properties of Eulerian digraphs.

▶ **Observation 2.1.** *Many Visits TSP has a tour of cost c if and only if there is a multiplicity function $m_G : V^2 \to \mathbb{Z}_{\geq 0}$ of cost c such that (i) for every $v \in V$, $\sum_{w \in V} m(v, w) = \sum_{w \in V} m(w, v) = k(v)$ and (ii) m contains a spanning connected subgraph.*

Thanks to Observation 2.1, in the remainder of this paper we refer to multiplicity functions as solutions of MVTSP (and some related problems which we are going to define). By standard arguments, the multiplicity function can be transformed to a tour in time $\mathcal{O}(\ell)$. Moreover, Grigoriev and Van de Klundert [14] describe an algorithm which transforms it to a compressed representation of the tour in time $O(n^4 \log \ell)$.

Out-trees. An *out-tree* is the digraph obtained from a rooted tree by orienting all edges away from the root. If an out-tree T is a subgraph of a directed graph G and additionally T spans the whole vertex set $V(G)$ we call T an *outbranching*. The sequence $\{\mathsf{outdeg}_T(v)\}_{v\in V(T)}$ is called the *outdegree sequence* of T. Consider a set of vertices $X \subseteq V$, $|X| \geq 2$.

▶ **Lemma 2.2** (Berger et al. [3], Lemma 2.4). *A sequence of nonnegative integers $\{d_v\}_{v\in X}$ is an outdegree sequence of an out-tree spanning X and rooted at $r \in X$ if and only if (i) $d_r \geq 1$ and (ii) $\sum_{v\in X} d_v = |X| - 1$.*

A sequence $\{d_v\}_{v\in X}$ that satisfies (i) and (ii) will be called an *out-tree sequence rooted at r*, or *outbranching sequence rooted at r* when additionally $X = V$. A δ-out-tree means any subtree spanning X with outdegree sequence δ.

3 Reduction to small demands

Consider the following problem, for a family of simple digraphs $\mathcal{F}$.

Fixed Degree $\mathcal{F}$-Subgraph
Input: $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$, $\mathsf{in}, \mathsf{out} : V \to \mathbb{Z}_{\geq 0}$
Question: Find a function $m : V^2 \to \mathbb{Z}_{\geq 0}$ such that
(i) G_m contains a member of $\mathcal{F}$ as a spanning subgraph,
(ii) for every $v \in V$ we have $\mathsf{in}(v) = \mathsf{indeg}_{G_m}(v)$ and $\mathsf{out}(v) = \mathsf{outdeg}_{G_m}(v)$, and
so as to minimize the value of $d(m) = \sum_{v,w\in V} d(v,w)m(v,w)$.

In this paper, we will consider two versions of the problem: when $\mathcal{F}$ is the family of all oriented trees, called Fixed Degree Connected Subgraph, and when $\mathcal{F}$ is the family of all out-trees with a fixed root r, called Fixed Degree Subgraph With Outbranching. The role of $\mathcal{F}$ is to force connectivity of the instance. Other choices for $\mathcal{F}$ can also be interesting, for example Cosmadakis and Papadimitriou [7] consider the family of minimal Eulerian digraphs.

The goal of this section is to show that, essentially, using a polynomial time preprocessing step we can reduce an instance of Fixed Degree $\mathcal{F}$-Subgraph to an equivalent one but with demands in, out bounded by $O(n^2)$.

When considering the instance of Fixed Degree $\mathcal{F}$-Subgraph we will use the notation $n = |V|$ and $\ell = \sum_{v\in V} \mathsf{in}(v)$. (Clearly, we can assume that also $\ell = \sum_{v\in V} \mathsf{out}(v)$, for otherwise there is no solution.)

Observe that if the image of d is $\{0, +\infty\}$ we get the natural unweighted version, where we are given a graph with edge set $d^{-1}(0)$ and the goal is to decide if one can choose multiplicities of the edges so that the resulting digraph contains a member of $\mathcal{F}$ and its in- and outdegrees match the demands of in and out.

The following observation follows by standard properties of Eulerian cycles in digraphs and the fact that every strongly connected graph contains an outbranching rooted at arbitrary vertex.

▶ **Observation 3.1.** *Many Visits TSP is a special case of both Fixed Degree Connected Subgraph and Fixed Degree Subgraph With Outbranching with $\mathsf{in}(v) = \mathsf{out}(v) = k(v)$ for every vertex $v \in V$.*

In the following lemma, we consider the relaxed problem Fixed Degree Subgraph, defined exactly as Fixed Degree $\mathcal{F}$-Subgraph, but dropping the constraint that solutions must contain a member of $\mathcal{F}$. In what follows, $s_n(\mathcal{F}) = \max_{G\in\mathcal{F},|V(G)|=n} |E(G)|$. (Note that in applications we consider in this work $\mathcal{F}$ is a family of oriented spanning trees, so $s_n(\mathcal{F}) = n - 1$.)

▶ **Lemma 3.2.** *Fix an input instance $d : V^2 \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$, $\mathsf{in}, \mathsf{out} : V^2 \to \mathbb{Z}_{\geq 0}$. For every optimal solution r of* Fixed Degree Subgraph *there is an optimal solution c' of* Fixed Degree $\mathcal{F}$-Subgraph *such that for every $u, v \in V$*

$$|r(u,v) - c'(u,v)| \leq s_{|V|}(\mathcal{F}).$$

Before we proceed to a formal proof of Lemma 3.2, let us sketch an intuition behind it. We pick an optimal solution c of Fixed Degree $\mathcal{F}$-Subgraph and let $B \in \mathcal{F}$ be a spanning subgraph in G_c. The symmetric difference between $E(G_r)$ and $E(G_c)$ can be decomposed into "alternating" cycles. It suffices to alternate $|E(B) \setminus E(G_r)| \leq s_n(\mathcal{F})$ of them to enforce containing B. If we alternated *all* the cycles, we would get the cost of exactly $d(c)$, but because of the optimality of r, alternating any of the cycles does not decrease the cost of the solution. Hence alternating a subset of them cannot make the cost bigger than $d(c)$.

Proof. Let c be an arbitrary optimal solution of Fixed Degree $\mathcal{F}$-Subgraph and let B be an arbitrary graph from $\mathcal{F}$ which is a spanning subgraph of G_c. Our plan is to build an optimal solution c' of Fixed Degree $\mathcal{F}$-Subgraph which contains B and does not differ too much from r.

Define multisets $A_c = E(G_c) \setminus E(G_r)$, $A_r = E(G_r) \setminus E(G_c)$ and $A = A_c \cup A_r = E(G_c) \triangle E(G_r)$. In what follows, by an *alternating cycle* we mean an even cardinality set of edges

$$\{(v_0, v_1), (v_2, v_1), (v_2, v_3), (v_4, v_3) \ldots, (v_{2\ell-2}, v_{2\ell-1}), (v_0, v_{2\ell-1})\},$$

where edges come alternately from A_c and A_r. Note that an alternating cycle is not really a directed cycle, it is just an orientation of a simple undirected cycle.

Note that for every vertex $v \in V$, among the edges in A that enter (resp. leave) v the number of edges from A_c is the same as the number of edges from A_r (counted with corresponding multiplicities), since both c and r satisfy the degree constraints for the same instance. It follows that A can be decomposed into a multiset $\mathcal{C}$ of alternating simple cycles, i.e.,

$$m_A = \sum_{C \in \mathcal{C}} m_C,$$

where $m_C : V^2 \to \mathbb{Z}_{\geq 0}$ and for each pair $u, v \in V$ we have $m_C(u,v) = [(u,v) \in C] \cdot m_{\mathcal{C}}(C)$. To clarify, we note that the sum above is over all cycles in $\mathcal{C}$, and not over all copies of cycles.

Denote $B^+ = E(B) \setminus E(G_r)$. Since $B^+ \subseteq A_c$, for each $e \in B^+$, there is at least one cycle in $\mathcal{C}$ that contains e. We choose an arbitrary such cycle and we denote it by C_e. (Note that it may happen that $C_e = C_{e'}$ for two different edges $e, e' \in B^+$.) Let $\mathcal{C}^+ = \{C_e \mid e \in B^+\}$. Then we define c', by putting for every $u, v \in V$

$$c'(u,v) = r(u,v) + (-1)^{[(u,v) \in A_r]} \sum_{C \in \mathcal{C}^+} [(u,v) \in C]. \tag{1}$$

In other words, c' is obtained from r by iterating over all cycles in $C \in \mathcal{C}^+$, and adding one copy of each edge of $C \cap A_c$ and removing one copy of each edge of $C \cap A_r$.

Let us show that $G_{c'}$ contains B. This is trivial for every $e \in B^+$. When $e \in E(B) \cap E(G_r)$, consider two cases. If $e \notin A_r$, then $c'(e) \geq r(e)$, so $e \in G_{c'}$. If $e \in A_r$, $m_A(e) = r(e) - c(e)$. Then $c'(e) = r(e) - \sum_{C \in \mathcal{C}^+} [(u,v) \in C] \geq r(e) - m_A(e) = c(e) \geq 1$, where the last inequality follows since $B \subseteq G_c$.

To see that c' satisfies the degree constraints, recall that r does so, and note that if in (1) we consider only the summands corresponding to a single cycle $C \in \mathcal{C}^+$, then for every vertex we either add one outgoing edge and remove one outgoing edge, or add one ingoing edge and remove one ingoing edge, or we do not change the set of edges incident to it.

For a cycle $C \in \mathcal{C}$ let $\delta(C) = d(A_c \cap C) - d(A_r \cap C)$. Observe that for every cycle $C \in \mathcal{C}$ we have $\delta(C) \geq 0$, for otherwise $E(G_r) \setminus (C \cap A_r) \cup (C \cap A_c)$ contradicts the optimality of r. It follows that

$$d(c') = d(r) + \sum_{C \in \mathcal{C}^+} \delta(C) \leq d(r) + \sum_{C \in \mathcal{C}} \delta(C) = d(c). \tag{2}$$

Hence, since c is optimal solution of Fixed Degree $\mathcal{F}$-Subgraph, we get that c' is optimal solution of Fixed Degree $\mathcal{F}$-Subgraph as well. Moreover, by (1), for every $u, v \in V$,

$$|c'(u,v) - r(u,v)| \leq |\mathcal{C}^+| \leq |B| \leq s_{|V|}(\mathcal{F}). \tag{3}$$

This ends the proof. ◀

As noted in [7, 2], Fixed Degree Subgraph can be solved by a reduction to minimum cost flow. By applying Orlin's algorithm [23] we get the following.

▶ **Observation 3.3** (Folklore, [7, 2]). *Fixed Degree Subgraph can be solved in time* $O(n^3 \log n)$.

▶ **Theorem 3.4** (Kernelization). *There is a polynomial time algorithm which, given an instance* $I = (d, \mathsf{in}, \mathsf{out})$ *of Fixed Degree* $\mathcal{F}$*-Subgraph, outputs an instance* $I' = (d, \mathsf{in}', \mathsf{out}')$ *of the same problem and a function* $f : V^2 \to \mathbb{Z}_{\geq 0}$ *such that*

(i) $\mathsf{in}'(v), \mathsf{out}'(v) = \mathcal{O}(n \cdot s_n(\mathcal{F}))$ *for every vertex* v,

(ii) *if* m^* *is an optimal solution for* I'*, then* $f + m^*$ *is an optimal solution for* I.

The algorithm does not need to know $\mathcal{F}$*, just the value of* $s_n(\mathcal{F})$.

Proof. Our algorithm begins by finding an optimal solution r of Fixed Degree Subgraph using Observation 3.3.

Define $f_0 : V^2 \to \mathbb{Z}_{\geq 0}$, where for every $v, w \in V$ we put $f_0(v,w) = \max\{r(v,w) - s_n(\mathcal{F}), 0\}$. By Lemma 3.2, there exists an optimal solution c' for instance I such that $c' \geq f_0$. Now define $f : V^2 \to \mathbb{Z}_{\geq 0}$, where for every $v, w \in V$ we put $f(v,w) = \max\{f_0(v,w) - 1, 0\}$. Finally, we put $\mathsf{in}'(v) = \mathsf{in}(v) - \sum_{w \in V} f(w,v)$ and $\mathsf{out}'(v) = \mathsf{out}(v) - \sum_{w \in V} f(v,w)$. The algorithm outputs $I' = (d, \mathsf{in}', \mathsf{out}')$ and f. In what follows, we show that the output has the desired properties.

For the property (i), consider any vertex $v \in V$ and observe that $\sum_{w \in V} f(v,w) \geq \sum_{w \in V}(f_0(v,w) - 1) \geq \sum_{w \in V}(r(v,w) - s_n(\mathcal{F}) - 1)$. Since r is a feasible solution of I, we have $\mathsf{out}(v) = \sum_{w \in V} r(v,w)$. It follows that $\mathsf{out}'(v) \leq n(1 + s_n(\mathcal{F})) = O(n \cdot s_n(\mathcal{F}))$ as required. The argument for $\mathsf{in}'(v)$ is symmetric.

Now we focus on (ii). Let m^* be an optimal solution for I'. It is easy to check that $f + m^*$ satisfies the degree constraints for the instance I. Also, since m^* contains a subgraph from $\mathcal{F}$, then $f + m^*$ contains the same subgraph. It follows that $f + m^*$ is a feasible solution of I. It suffices to show that $f + m^*$ is an optimal solution for I.

Denote $r = c' - f$. Consider any pair $v, w \in V$ such that $c'(v,w) \geq 1$. We claim that $f(v,w) \leq c'(v,w) - 1$. Indeed, if $f_0(v,w) = 0$ then $f(v,w) = 0 \leq c'(v,w) - 1$, and if $f_0(v,w) \geq 1$ then $f(v,w) = f_0(v,w) - 1 \leq c'(v,w) - 1$. It follows that $r(v,w) \geq 1$. In particular, since c' contains a subgraph from $\mathcal{F}$, then also r contains the same subgraph. It follows that r is a feasible solution for I' (the degree constraints are easy to check). Hence, $d(m^*) \leq d(r)$. It follows that $d(f + m^*) \leq d(r + f) = d(c')$, so $f + m^*$ is indeed an optimal solution for I. ◀

4 The small costs case in time $\mathcal{O}^*(2^n D)$

In this section we establish Theorem 1.1. We do it in a bottom-up fashion, starting with a simplified core problem, and next generalizing the solution in a few steps.

4.1 Unweighted decision version with small degree demands

Consider the following problem.

Decision Unweighted Fixed Degree Connected Subgraph
Input: a digraph $G = (V, E)$, $\mathsf{in}, \mathsf{out} : V \to \mathbb{Z}_{\geq 0}$
Question: Is there a function $m : V^2 \to \mathbb{Z}_{\geq 0}$ such that $G_m^{\downarrow}$ is a connected subgraph of G and for every $v \in V$ we have $\mathsf{in}(v) = \mathsf{indeg}_{G_m}(v)$ and $\mathsf{out}(v) = \mathsf{outdeg}_{G_m}(v)$?

Note that Decision Unweighted Fixed Degree Connected Subgraph generalizes the directed Hamiltonian cycle problem, which is known to be solvable in $\mathcal{O}^*(2^n)$ time and polynomial space. In this section we show that this running time can be obtained for the more general problem as well, though we need to allow some randomization.

▶ **Theorem 4.1.** *There is a randomized algorithm which solves an instance $I = (\mathsf{in}, \mathsf{out})$ of Decision Unweighted Fixed Degree Connected Subgraph in time $\mathcal{O}^*(2^n \operatorname{poly}(M))$ and polynomial space, where $M = \max_v \max\{\mathsf{in}(v), \mathsf{out}(v)\}$. The algorithm is Monte Carlo with one-sided error, i.e., the positive answer is always correct and the negative answer is correct with probability at least p, for any constant $p < 1$.*

Our strategy will be to reduce our problem to detecting a perfect matching in a bipartite graph with an additional connectivity constraint.

We define a bipartite graph $B_G = (O, I, E(B_G))$ as follows. Let $I = \{v_1^I, \ldots, v_{\mathsf{in}(v)}^I \mid v \in V(H)\}$, $O = \{v_1^O, \ldots, v_{\mathsf{out}(v)}^O \mid v \in V(H)\}$, and $E(B_G) = \{u_i^O v_j^I \mid (u, v) \in E(G)\}$.

▶ **Observation 4.2.** *$|I| = |O| = \mathcal{O}(nM)$ and $|E(B_G)| \leq E(G)M^2 = \mathcal{O}(n^2M^2)$.*

For an undirected graph H by $\mathcal{PM}(H)$ we denote the set of perfect matchings in H. We say that a matching M in B_G is *connected* when for every cut $(X, V \setminus X)$ with $\emptyset \neq X \subsetneq V$ the matching M contains an edge $u_i^O v_j^I$ such that $u \in X$ and $v \in V \setminus X$ or $v \in X$ and $u \in V \setminus X$.

For a matching M in B_G we define a *contraction* of M as function $m : V^2 \to \mathbb{Z}_{\geq 0}$ such that $m(u, v) = |\{u_i^O v_j^I \in M \mid i \in [\mathsf{out}(u)], j \in [\mathsf{in}(v)]\}|$. In other words G_m is obtained from M by (1) orienting every edge from O to I and (2) identifying all vertices in $\{v_1^I, \ldots, v_{\mathsf{in}(v)}^I\} \cup \{v_1^O, \ldots, v_{\mathsf{out}(v)}^O\}$ for every $v \in V$, and keeping the multiple edges and loops.

▶ **Lemma 4.3.** *$(G, \mathsf{in}, \mathsf{out})$ is a yes-instance of Decision Unweighted Fixed Degree Connected Subgraph iff graph B_G contains a connected perfect matching.*

Proof. Let M be a connected perfect matching in B_G and let m be its contraction. We claim that m is a solution of $(G, \mathsf{in}, \mathsf{out})$. By the definition of B_G, $G_m^{\downarrow}$ is a subgraph of G. Since M is connected, G_m is connected as well, and so is $G_m^{\downarrow}$. Moreover, since M is a perfect matching $\mathsf{in}(v) = \mathsf{indeg}_{G_m}(v)$ and $\mathsf{out}(v) = \mathsf{outdeg}_{G_m}(v)$ for every vertex v.

For the other direction, let m be a solution for $(G, \mathsf{in}, \mathsf{out})$. For every $v \in V$, there are exactly $\mathsf{out}(v)$ edges leaving v in G_m. Let us denote them $e_{v,1}^O, \ldots, e_{v,\mathsf{out}(v)}^O$. Similarly, let us denote all the edges entering v by $e_{v,1}^I, \ldots, e_{v,\mathsf{in}(v)}^I$. Then we define M as the set of edges of the form $u_i^O v_j^I$ such that G_m contains an edge $e = e_{u,i}^O = e_{v,j}^I$. The fact that M is a perfect matching is clear from the construction. Also, M is connected, for otherwise G_m is not connected. ◀

From now on, let $B = (O, I, E(B))$ be an arbitrary subgraph of B_G. Define the following multivariate polynomial over $\mathrm{GF}(2^t)$, for an integer t to be specified later.

$$R = \sum_{\substack{M \in \mathcal{PM}(B) \\ M \text{ is connected}}} \prod_{e \in M} x_e \tag{4}$$

▶ **Lemma 4.4.** *R is not the zero polynomial if and only if B contains a connected perfect matching.*

Proof. It is clear that if R is non-zero then B contains a connected perfect matching. For the reverse implication it suffices to notice that every summand in R has a different set of variables, so it does not cancel out with other summands over $\mathrm{GF}(2^t)$. ◀

Our strategy is to test whether R is non-zero by means of DeMillo–Lipton–Schwartz–Zippel Lemma, which we recall below.

▶ **Lemma 4.5** (DeMillo and Lipton [11], Schwartz [25], Zippel [28])**.** *Let $P(x_1, x_2, \ldots, x_m)$ be a nonzero polynomial of degree at most d over a field $\mathbb{F}$ and let S be a finite subset of $\mathbb{F}$. Then, the probability that P evaluates to zero on a random element $(a_1, a_2, \ldots, a_m) \in S^m$ is bounded by $d/|S|$.*

By Lemmas 4.4 and 4.5, the task reduces to *evaluating* R fast. To this end, we will define a different polynomial P which is easier to evaluate and turns out to be equal to R over $\mathrm{GF}(2^t)$.

Consider a subset $X \subseteq V$. Let $I_X = \{v_i^I \in I \mid v \in X, i = 1, \ldots, \mathsf{in}(v)\}$ and $O_X = \{v_i^O \in O \mid v \in X, i = 1, \ldots, \mathsf{out}(v)\}$. Abusing the notation slightly, we will denote $B[X] = B[I_X \cup O_X]$. Define the following polynomial.

$$P_X = \sum_{M \in \mathcal{PM}(B[X])} \prod_{e \in M} x_e \tag{5}$$

In what follows, v^* is an arbitrary but fixed vertex of V. Define yet another polynomial.

$$P = \sum_{\substack{X \subseteq V \\ v^* \in X}} P_X P_{V \setminus X}. \tag{6}$$

▶ **Lemma 4.6.** *$P = R$.*

Proof. For a matching M in B we say that a set $X \subseteq V$ is *consistent* with M when M does not contain an edge $u_i^O v_j^I$ such that $u \in X$ and $v \in V \setminus X$ or $v \in X$ and $u \in V \setminus X$. The family of all subsets of V that are consistent with M will be denoted by $\mathcal{C}(M)$. Then we can rewrite P as follows.

$$\begin{aligned}
P &= \sum_{\substack{X \subseteq V \\ v^* \in X}} \sum_{M_1 \in \mathcal{PM}(B[X])} \sum_{M_2 \in \mathcal{PM}(B[V \setminus X])} \prod_{e \in M_1 \cup M_2} x_e && \text{[definition]} \\
&= \sum_{M \in \mathcal{PM}(B)} \sum_{\substack{X \in \mathcal{C}(M) \\ v^* \in X}} \prod_{e \in M} x_e && \text{[group by } M = M_1 \uplus M_2\text{]} \\
&= \sum_{M \in \mathcal{PM}(B)} |\{X \in \mathcal{C}(M) \mid v^* \in X\}| \prod_{e \in M} x_e && \text{[trivial]}
\end{aligned}$$

Let us consider a perfect matching $M \in \mathcal{PM}(B)$ and the corresponding contraction m. Observe that the number of sets that are consistent with M and contain a vertex v^* is equal to $2^{\mathrm{cc}(M)-1}$, where $\mathrm{cc}(M)$ is the number of connected components of G_m. Indeed, when X is consistent with M, then for every connected component Q of G_m, either $V(Q) \subseteq X$ or $V(Q) \subseteq V \setminus X$. For the component that contains v^* the choice is fixed, while every choice for the remaining components defines a set consistent with M. It follows that when M is not connected $\mathrm{cc}(M) \geq 2$, and the value of $2^{\mathrm{cc}(M)-1}$ is equal to 0 in $\mathrm{GF}(2^t)$, so the corresponding summand vanishes. On the other hand, if M is connected, the corresponding summand equals just $\prod_{e \in M} x_e$ and it does not cancel out with another summand because the monomial has a unique set of variables. It follows that $P = R$. ◀

▶ **Lemma 4.7** (Tutte, Lovász [26, 22])**.** *For an arbitrary set $X \subseteq V$, the polynomial P_X can be evaluated using $\mathrm{poly}(n+M)$ field operations.*

Proof. Compute the determinant of the corresponding Tutte matrix of dimension $|O| \times |I|$. ◀

Let us now fix our field, namely $t = \lceil 1 + \log n + \log M \rceil$. Since arithmetic operations in $\mathrm{GF}(2^t)$ can be performed in time $\mathcal{O}(t \log^2 t) = \mathcal{O}(\log(n+M) \log^2 \log(n+m))$, by the definition of P and Lemma 4.7 we get the following corollary.

▶ **Corollary 4.8.** *P can be evaluated in time $2^n \,\mathrm{poly}(n+M)$.*

▶ **Lemma 4.9.** *There is a randomized algorithm which decides if B contains a connected perfect matching in time $\mathcal{O}^*(2^n \,\mathrm{poly}(M))$ and polynomial space, where $M = \max_v \max\{\mathsf{in}(v), \mathsf{out}(v)\}$. The algorithm is Monte Carlo with one-sided error, i.e., the positive answer is always correct and the negative answer is correct with probability at least p, for any constant $p < 1$.*

Proof. The algorithm evaluates polynomial P using Corollary 4.8 substituting a random element of $\mathrm{GF}(2^t)$ for each variable, and reports "yes" when the evaluation is nonzero and "no" otherwise. If it reported "yes", then P was a non-zero polynomial and by Lemma 4.4 the answer is correct. Assume it reported 'no' for a yes-instance. By Lemma 4.4 P is non-zero. Since $\deg P = |I| \leq nM$, by Lemma 4.5 the probability that P evaluated to 0 is bounded by $\deg P / 2^t \leq 1/2$ and we can make this probability arbitrarily small by repeating the whole algorithm a number of times, and reporting "yes" if at least one evaluation was nonzero. The claim follows. ◀

Theorem 4.1 follows immediately from Lemma 4.3 and Lemma 4.9 applied to B_G.

4.2 Finding the solution

▶ **Lemma 4.10.** *There is a randomized algorithm which, given a yes-instance of* Decision Unweighted Fixed Degree Connected Subgraph*, always returns the corresponding solution m in expected time $\mathcal{O}^*(2^n \,\mathrm{poly}(M))$. The time can be made deterministic at the cost of introducing arbitrarily small probability of failure.*

In order to prove Lemma 4.10 we cast the problem in the setting of *inclusion oracles* from the work of Björklund et al. [5]. Consider a universe U and an (unknown) family of *witnesses* $\mathcal{F} \subseteq 2^U$. An *inclusion oracle* is a procedure which, given a query set $Y \subseteq U$, answers (either YES or NO) whether there exists at least one witness $W \in \mathcal{F}$ such that $W \subseteq Y$. Björklund et al. prove the following.

▶ **Theorem 4.11** ([5]). *There exists an algorithm that extracts a witness of size k in $\mathcal{F}$ using in expectation at most $O(k \log |U|)$ queries to a randomized inclusion oracle that has no false positives but may output a false negative with probability at most $p \leq \frac{1}{4}$.*

Proof of Lemma 4.10. Let $U = E(B_G)$ and let $\mathcal{F}$ be the family of all connected perfect matchings in B_G. Note that $|U| = O(n^2M^2)$ and witnesses in $\mathcal{F}$ have all size $|I| = O(nM)$. Then, Lemma 4.9 provides a randomized inclusion oracle and we can apply Theorem 4.11. (If one insists on deterministic, and not expected, running time, it suffices to chose a sufficiently large constant r and stop the algorithm if it exceeds the expected running time at least r times – by Markov's inequality, this happens with probability at most $1/r$.) ◀

4.3 Proof of Theorem 1.1

In the lemma below we will adapt the construction from Section 4.1 to the weighted case in a standard way, by introducing a new variable tracking the weight.

▶ **Lemma 4.12.** *There is a randomized algorithm which solves an instance $I = (d, \mathsf{in}, \mathsf{out}, w)$ of Fixed Degree Connected Subgraph in time $\mathcal{O}^*(2^n D \operatorname{poly}(M))$ and polynomial space, where $M = \max_v \max\{\mathsf{in}(v), \mathsf{out}(v)\}$ and D is the maximum integer value of d. The algorithm returns a minimum weight solution with probability at least p, for any constant $p < 1$.*

Proof. Define $G = (V, E)$ where $E = \{(u, v) \in V^2 \mid d(u, v) \in \mathbb{Z}_{\geq 0}\}$. Let R' be the polynomial obtained from R by replacing every variable x_e for $e = u_i^O v_j^I \in E(B_G)$ by the product $x_e \cdot y^{d(u,v)}$, where y is a new variable. Proceed similarly with P, obtaining P'. By Lemma 4.4, $P' = R'$. Decompose R' as $R' = \sum_{i=0}^{|I|\cdot D} R'_i y^i$, where R'_i, for every $i = 0, \ldots, |I| \cdot D$, is a polynomial in variables $\{x_e\}_{e \in E(B_G)}$. The monomials in R'_i enumerate all matchings M such that the contraction m of M has weight $d(m) = i$. By the construction in the proof of Lemma 4.3 R'_i is non-zero if and only if instance I has a solution of weight i. Using Lagrange interpolation, we can recover the value of each R'_i for random values of the variables $\{x_e\}_{e \in E(B_G)}$ (the values are the same for all the polynomials). The interpolation algorithm requires $|I| \cdot D = \mathcal{O}(nMD)$ evaluations of R'. Since $R' = P'$, by Lemma 4.8 each of them takes $2^n \operatorname{poly}(n + M))$ time. Our algorithm reports the minimum w such that R'_w evaluated to a non-zero element of $\mathrm{GF}(2^t)$, or $+\infty$ if no such w exists. The solution of weight w is then found using Lemma 4.10. The event that the optimum value w^* is not reported means that R'_{w^*} is a non-zero polynomial that evaluated to 0 at the randomly chosen values. By Lemma 4.5 this happens with probability at most $\deg P / 2^t \leq 1/2$, and one can make this probability arbitrarily small by standard methods. ◀

Theorem 1.1 follows now immediately by applying Theorem 3.4 which reduces the general problem to the $M = \mathcal{O}(n^2)$ case and solving the resulting instance by Lemma 4.12. Theorem 1.1 says in particular that if finite weights are bounded by a polynomial in n then we can solve Many Visits TSP in time $\mathcal{O}^*(2^n)$ and polynomial space by a randomized algorithm with no false positives and with false negatives with arbitrarily small constant probability.

5 The general case

In this section we prove Theorem 1.3, i.e., we show an algorithm solving Many Visits TSP in time $\mathcal{O}^*(4^n)$. In fact, we do not introduce a new algorithm, but we consider an algorithm by Berger et al. (Algorithm 5 in [3]) and we provide a refined analysis, resulting in an improved running time bound $\mathcal{O}^*(4^n)$, which is tight up to a polynomial factor.

Let us recall the algorithm of Berger et al., in a slightly changed notation. In fact, they solve a slightly more general problem, namely Fixed Degree Subgraph With Outbranching. Let $I = (d, \mathsf{in}, \mathsf{out}, r)$ be an instance of this problem, i.e., we want to find a solution $m : V^2 \to \mathbb{Z}_{\geq 0}$ that satisfies the degree constraints specified by in and out and contains an outbranching rooted at r. In what follows we assume $V = \{1, \ldots, n\}$ and $r = 1$.

Consider an outbranching sequence $\{\delta_v\}_{v \in V}$ rooted at $r = 1$. In what follows, all outbranching sequences will be rooted at 1, so we skip specifying the root. Let T_δ be a minimum cost outbranching among all outbranchings with outdegree sequence δ and let r_δ be an optimum solution of Fixed Degree Subgraph for instance $(d, \mathsf{in}', \mathsf{out}')$ where $\mathsf{out}' = \mathsf{out} - \mathsf{outdeg}_T$ and $\mathsf{in}' = \mathsf{in} - \mathsf{indeg}_T$. Berger et al. note that then $m_\delta = m_{T_\delta} + r_\delta$ is a feasible solution for instance I of Fixed Degree Subgraph With Outbranching, and moreover it has minimum cost among all solutions that contain an outbranching with outdegree sequence δ. Since r_δ can be found in polynomial time by Observation 3.3, in order to solve instance I it suffices to find outbranchings T_δ for all outbranching sequences δ and return the solution m_δ of minimum cost. Hence, Theorem 1.3 boils down to proving the following lemma.

▶ **Lemma 5.1.** *There is an algorithm which, for every outbranching sequence δ, finds a minimum cost outbranching among all outbranchings with outdegree sequence δ and runs in time $\mathcal{O}^*(4^n)$.*

We prove Lemma 5.1 by using dynamic programming (DP). However, it will be convenient to present the DP as a recursive function BestOutbranching with two parameters, $S \subseteq V$ and $\{\delta_v\}_{v \in S}$ (see Algorithm 1). It is assumed that $1 \in S$. We will show that BestOutbranching(S, δ) returns a minimum cost out-tree among all out-trees with outdegree sequence δ that are rooted at 1 and span S. Our algorithm runs BestOutbranching for $S = V$ and all outbranching sequences $\delta : V \to \mathbb{Z}_{\geq 0}$. Whenever BestOutbranching returns a solution for an input (S, δ), it is memoized (say, in an efficient dictionary), so that when BestOutbranching is called with parameters (S, δ) again, the output can be retrieved in polynomial time.

Algorithm 1 A pseudocode of the algorithm from Lemma 5.1.

```
function BestOutbranching(S, δ)
    v_first ← min{v ∈ S | δ_v = 0}
    if |S| = 2 then return {(1, v_first)}.
    else
        minCost ← ∞
        for w ∈ S do
            if (δ_w ≥ 1 ∧ w ≠ 1) ∨ (δ_w ≥ 2 ∧ w = 1) then
                S' ← S \ {v_first}
                δ' ← δ|_{S'}
                δ'_w ← δ'_w − 1
                R_w ← BestOutbranching(S', δ') ∪ {(w, v_first)}
                if d(R_w) < minCost then
                    minCost ← d(R_w)
                    best ← R_w
        return best
```

Let us define $\mathsf{lastRmvd}(S) := \max(\{0, 1, 2, \ldots, n\} \setminus S)$ and $\mathsf{bad}(S, \delta) := \{v \in S \mid v < \mathsf{lastRmvd}(S) \wedge \delta_v = 0\}$. Let us call (S, δ) a *reachable state* if it meets the following conditions:

(i) $\delta_1 \geq 1$
(ii) $\sum_{v \in S} \delta_v = |S| - 1$
(iii) $|\mathsf{bad}(S, \delta)| \leq 1$

▶ **Lemma 5.2.** *If function* BestOutbranching *is given a reachable state as input then all recursively called* BestOutbranching *will also be given only reachable states.*

Proof. Let us fix a reachable state (S, δ) for $|S| > 2$ and consider the associated value v_{first} from the algorithm. Denote $S' = S \setminus \{v_{\mathsf{first}}\}$. Clearly, it suffices to show that all pairs (S', δ') created in the **for** loop are reachable states. First, let us argue that $\mathsf{bad}(S', \delta) = \emptyset$. There are two cases:

- Assume $|\mathsf{bad}(S, \delta)| = 0$. In this case $v_{\mathsf{first}} > \mathsf{lastRmvd}(S)$ so $\mathsf{lastRmvd}(S') = v_{\mathsf{first}}$. Then, $\mathsf{bad}(S', \delta) = \{v \in S' \mid v < \mathsf{lastRmvd}(S') \wedge \delta_v = 0\} = \{v \in S \mid v < v_{\mathsf{first}} \wedge \delta_v = 0\} = \emptyset$.
- Assume $|\mathsf{bad}(S, \delta)| = 1$. Then, (1) $\mathsf{lastRmvd}(S') = \mathsf{lastRmvd}(S)$ because $\mathsf{lastRmvd}(S) > v_{\mathsf{first}}$ and (2) $\mathsf{bad}(S, \delta) = \{v_{\mathsf{first}}\}$. It follows that $\mathsf{bad}(S', \delta) \overset{(1)}{=} \{v \in S' \mid v < \mathsf{lastRmvd}(S) \wedge \delta_v = 0\} = \mathsf{bad}(S, \delta) \setminus \{v_{\mathsf{first}}\} \overset{(2)}{=} \emptyset$.

Let us consider the recursive call of BestOutbranching for a particular w. The sequence $\delta'|_{S'}$ differs from δ only at w, so $\mathsf{bad}(S', \delta') \subseteq \{w\} \cup \mathsf{bad}(S', \delta) = \{w\}$. This means that condition (iii) from the definition of a reachable state holds for (S', δ'). Since (S, δ) is reachable, $\delta_1 \geq 1$. Then either $w \neq 1$ and $\delta'_1 = \delta_1 \geq 1$ or $w = 1$ and $\delta'_1 = \delta_1 - 1 \geq 1$, where the last inequality holds thanks to the condition in the **if** statement in Algorithm 1. In both cases, (i) holds for (S', δ'). Finally, (ii) is immediate by the definition of δ'. It follows that (S', δ') is a reachable state, as required. ◀

▶ **Lemma 5.3.** *If the function* BestOutbranching *is given a reachable state (S, δ), it returns a cheapest out-tree T rooted at vertex 1, spanning S and with outdegree sequence δ.*

Proof. We will use induction on $|S|$.

In the base case $|S| = 2$, there is only one outbranching spanning S rooted at 1, namely $\{(1, v_{\mathsf{first}})\}$ and it is indeed returned by the algorithm.

In the inductive step assume $|S| > 2$. By conditions (i) and (ii) in the definition of a reachable state and Lemma 2.2, there is at least one out-tree rooted at 1, spanning S, and with outdegree sequence δ. Let T be a cheapest out-tree among all such out-trees. Vertex v_{first} is a leaf of T, since $\delta_{v_{\mathsf{first}}} = 0$. At some point w in the **for** loop in Algorithm 1 is equal to the parent w^* of v_{first} in T. Then, $T \setminus \{(w^*, v_{\mathsf{first}})\}$ is an out-tree rooted at 1, spanning S', and with outdegree sequence δ'. Since (S', δ') is a reachable state by Lemma 5.2, by the inductive hypothesis we know that a cheapest such out-tree T' will be returned by BestOutbranching(S', δ'). In particular, it means that $d(T') \leq d(T \setminus \{(w^*, v_{\mathsf{first}})\})$. Denote $R_{w^*} = T' \cup \{(w^*, v_{\mathsf{first}})\}$. Then, $d(R_{w^*}) = d(T') + d(w^*, v_{\mathsf{first}}) \leq d(T \setminus \{(w^*, v_{\mathsf{first}})\}) + d(w^*, v_{\mathsf{first}}) = d(T)$. It follows that BestOutbranching returns a set of edges best of cost at most $d(T)$. However $\mathsf{best} = R_w$ for a vertex w and by applying the induction hypothesis it is easy to see that R_w is an out-tree rooted at 1, spanning S with outdegree sequence δ. The claim follows. ◀

▶ **Lemma 5.4.** *There are $\mathcal{O}^*(4^n)$ reachable states.*

Proof. Any sequence of n nonnegative integers that sums up to at most $n-1$ will be called an *extended sequence*. It is well known that there are exactly $\binom{2n-1}{n} < 2^{2n-1} = \mathcal{O}(4^n)$ such sequences. To see this consider sequences of $n-1$ balls and n barriers and bijectively map them to the sequences of n numbers by counting balls between barriers and discarding the balls after the last barrier.

Let us fix an extended sequence $\bar{\delta} = \{\bar{\delta}_v\}_{v \in V}$, and denote $\bar{s} := n - (1 + \sum_{i=1}^{n} \bar{\delta}_i)$. We claim that there are only $\mathcal{O}(n)$ reachable states (S, δ) such that $\bar{\delta}|_S = \delta$ and $\bar{\delta}|_{V \setminus S} = 0$. Consider any such pair (S, δ). Let $(v_1, v_2, \ldots, v_k)$ be the vertices of $\{v \in V \mid \bar{\delta}_v = 0\}$ sorted

in increasing order. By the definition of a reachable state we know that $|S| = 1 + \sum_{i=1}^{n} \bar{\delta}_i$, so $\bar{s} = |\{1, 2, \ldots, n\} \setminus S|$. By (ii), for at least one vertex $v \in S$ we have $\bar{\delta}_v = \delta_v = 0$, so $k \geq \bar{s} + 1$. Let us assume that $k \geq \bar{s} + 2$ and $\mathsf{lastRmvd}(S) \geq v_{\bar{s}+2}$. Then, $\{v_1, v_2, \ldots, v_{\bar{s}+1}\} \cap S \subseteq \mathsf{bad}(S, \delta)$. Since $v_{\bar{s}+2} \leq \mathsf{lastRmvd}(S) \notin S$, at most $\bar{s} - 1$ elements from $\{v_1, v_2, \ldots, v_{\bar{s}+1}\}$ are outside S, so $|\mathsf{bad}(S, \delta)| \geq (\bar{s} + 1) - (\bar{s} - 1) = 2$. This is a contradiction with (S, δ) being a reachable state, which proves that $k \leq \bar{s} + 1$ or $\mathsf{lastRmvd}(S) < v_{\bar{s}+2}$. In any case, $\{1, 2, \ldots, n\} \setminus S \subseteq \{v_1, \ldots, v_{\bar{s}+1}\}$. There are $\bar{s} + 1 = \mathcal{O}(n)$ ways to choose $\bar{s}$ elements to the set $\{1, 2, \ldots, n\} \setminus S$ from $\{v_1, \ldots, v_{\bar{s}+1}\}$, so equivalently there are $\mathcal{O}(n)$ sets S such that (S, δ) is a reachable state, $\bar{\delta}|_S = \delta$ and $\bar{\delta}|_{V \setminus S} = 0$.

Every reachable state (S, δ) has the corresponding extended sequence $\{\bar{\delta}\}_{v \in V}$ defined by $\bar{\delta}|_S = \delta$ and $\bar{\delta}|_{V \setminus S} = 0$. Since there are $\mathcal{O}(4^n)$ extended sequences, and each of them has $O(n)$ corresponding reachable states there are $\mathcal{O}(4^n) \cdot \mathcal{O}(n) = \mathcal{O}^*(4^n)$ reachable states in total. ◀

We are ready to prove Lemma 5.1. Recall that our algorithm runs BestOutbranching(V, δ) for all outbranching sequences δ and uses memoization to avoid repeated computation. We claim that for any outbranching sequence δ, the pair (V, δ) is a reachable state. Indeed, conditions (i) and (ii) hold since δ is an outbranching sequence. By definition, $\mathsf{lastRmvd}(V) = 0$, so $\mathsf{bad}(V, \delta) = \emptyset$ which implies (iii). Hence by Lemma 5.3 the algorithm is correct. By Lemma 5.2 the running time can be bounded by the number of reachable states times a polynomial, which is $\mathcal{O}^*(4^n)$ by Lemma 5.4. This ends the proof of Lemma 5.1 and hence also Theorem 1.3, as discussed in the beginning of this section.

6 Polynomial space

In this section we show Theorem 1.4, that is, we solve Many Visits TSP in $\mathcal{O}^*(7.88^n)$ time and polynomial space. Berger et al. [2] solved this problem in $\mathcal{O}(16^{n+o(n)})$ time and polynomial space, with the key ingredient being the following.

▶ **Lemma 6.1** (Berger et al. [2]). *There is a polynomial space algorithm running in time $\mathcal{O}(4^{n+o(n)})$ which, given an outdegree sequence $\{\delta_v\}_{v \in V}$, a cost function $d : V^2 \to \mathbb{Z}_{\geq 0}$, and a root $r \in V$ computes the cheapest outbranching rooted at r with the required outdegrees.*

More precisely, the $\mathcal{O}(16^{n+o(n)})$-time algorithm consists of the following steps:

- **(i)** Enumerate all $\mathcal{O}(4^n)$ outbranching sequences
- **(ii)** For each outbranching sequence compute the cheapest outbranching with required degrees using Lemma 6.1 in time $\mathcal{O}(4^{n+o(n)})$
- **(iii)** For each of these outbranchings complete it to a solution of the original Many Visits TSP instance with an optimal solution of Fixed Degree Subgraph on the residual degree sequences (in polynomial time, by Observation 3.3).

The intuition behind our approach is as follows. We iterate over all subsets of vertices R. Here, R represents our guess of the set of inner vertices of an outbranching in an optimal solution. Then we perform (i) and (ii) in the smaller subgraph induced by R. Finally, we replace (iii) by a more powerful flow-based algorithm which connects the vertices in $V \setminus R$ to R, and at the same time computes a feasible solution of Fixed Degree Subgraph on the residual degree sequences, so that the total cost is minimized. Let $r = |R|$. Clearly, when r is a small fraction of n, we get significant savings in the running time. The closer r/n is to 1 the smaller are the savings, but also the smaller is the number $\binom{n}{r}$ of sets R to examine.

In fact, the real algorithm is slightly more complicated. Namely, we fix an integer parameter K, and then R corresponds to the set of vertices left from an outbranching in an optimal solution after K iterations of removing all leaves. The running time of our algorithm depends on K, because the algorithm actually guesses the layers of leaves in each iteration. The space complexity is polynomial and does not depend on K. In the end of this section, we show that our running time bound is minimized when $K = 4$.

6.1 Our algorithm

Similarly as in Section 5, we solve the more general Fixed Degree Subgraph With Outbranching: for a given instance $I = (d, \mathsf{in}, \mathsf{out}, \mathsf{root})$ we want to find a solution $m : V^2 \to \mathbb{Z}_{\geq 0}$ that satisfies the degree constraints specified by in and out and contains an outbranching rooted at root.

Let T be an arbitrary outbranching. We define a sequence $L_1(T), L_2(T), \ldots$ of subsets of $V(T)$ as follows. For $i \geq 1$ let $L_i(T)$ be the set of leaves of $T \setminus (L_1(T) \cup L_2(T) \cup \ldots \cup L_{i-1}(T))$ if $|V(T) \setminus (L_1(T) \cup \ldots \cup L_{i-1}(T))| > 1$, and otherwise $L_i = \emptyset$. The sets $L_i(T)$ will be called *leaf layers*. Denote $R_i(T) = V \setminus (L_1(T) \cup \cdots \cup L_i(T))$ for any $i \geq 1$.

▶ **Lemma 6.2.** *For every $i \geq 1$ we have $\mathsf{root} \in R_i(T) \setminus L_{i+1}(T)$, $|L_i(T)| \geq |L_{i+1}(T)|$ and $|L_{i+1}| \leq \frac{n-|R_i(T)|}{i}$.*

Proof. In this proof we skip the "(T)" in L_i and R_i because there is no ambiguity. Assume $\mathsf{root} \in L_i$ for some $i \geq 1$. It means that root is a leaf in $T \setminus (L_1 \cup L_2 \cup \ldots \cup L_{i-1})$. Then $V \setminus (L_1 \cup L_2 \cup \ldots \cup L_{i-1}) = \{\mathsf{root}\}$ and $L_i = \emptyset$, a contradiction. Hence $\mathsf{root} \notin L_i$ for all $i \geq 1$, and in consequence $\mathsf{root} \in R_i$ for all $i \geq 1$. However, $\mathsf{root} \in R_{i+1}(T)$ implies that $\mathsf{root} \notin L_{i+1}(T)$, hence $\mathsf{root} \in R_i \setminus L_{i+1}$.

If $|V \setminus (L_1 \cup \ldots \cup L_i)| > 1$, then L_{i+1} is the set of leaves of the out-tree $T \setminus (L_1 \cup \ldots \cup L_i)$, which is contained in the set of parents of vertices in L_i. Since every vertex in L_i has exactly one parent, $|L_i| \geq |L_{i+1}|$. If $|V \setminus (L_1 \cup \ldots \cup L_i)| \leq 1$ then $L_{i+1} = \emptyset$ and clearly $|L_i| \geq |L_{i+1}| = 0$.

Finally, since for every $j < i$ we have $|L_j| \geq |L_i|$ we get $n - |R_i| = |L_1| + \ldots + |L_i| \geq i|L_i|$. It follows that $|L_{i+1}| \leq |L_i| \leq \frac{n-|R_i|}{i}$, as required. ◀

Pseudocode of our algorithm is presented as Algorithm 2.

Algorithm 2 A pseudocode of the algorithm from Section 6.1.

```
1:  function Solve(G, out, in, d, root)
2:      best ← ∞
3:      for R, L_{K+1}, δ do
4:          T_R ← cheapest δ-out-tree spanning R rooted at root (Lemma 6.1)
5:          out' ← out − outdeg_{T_R}
6:          in' ← in − indeg_{T_R}
7:          for L_1, ..., L_K do
8:              F ← CreateNetwork(G, R, out', in', d, L_1, ..., L_K)
9:              f ← MinCostMaxFlow(F)
10:             if |f| = Σ_{v∈V(G)} out'(v) and cost(f) + d(T_R) < best then
11:                 best ← cost(f) + d(T_R)
        return best
```

For clarity, in the pseudocode we skipped some constraints that we enforce on the sets L_i and sequence δ. We state them below.

(C1) $L_{K+1} \subseteq R \subseteq V, \mathsf{root} \in R \setminus L_{K+1}, |L_{K+1}| \leq \frac{n-|R|}{K}$

(C2) $\{\delta_v\}_{v \in R}$ is a rooted out-tree sequence, i.e., for all $v \in R$ we have $\delta_v \in \mathbb{Z}_{\geq 0}, \sum_{v \in R} \delta(v) = |R| - 1$; also $\delta_{\mathsf{root}} \geq 1$ if $|R| \geq 2$ and $\delta_{\mathsf{root}} = 0$ if $|R| = 1$.

(C3) for every $v \in L_{K+1}$ we have $\delta_v = 0$ and for every $v \in R \setminus (L_{K+1} \cup \{\mathsf{root}\})$ we have $\delta_v \geq 1$

(C4) $L_1 \uplus L_2 \uplus \cdots \uplus L_K = V \setminus R$

(C5) $|L_i| \geq |L_{i+1}|$ for $i = 1, \ldots, K$.

It is clear that all these possibilities can be enumerated in time proportional to their total number times $O(n)$.

Let us provide some further intuition about Algorithm 2. Consider an optimum solution m of I and any outbranching B in m rooted at root. In Algorithm 2, for any $i = 1, \ldots K+1$, the set L_i is a guess of the leaf layer $L_i(B)$, while R is a guess of $V \setminus (L_1(B) \cup \cdots \cup L_K(B))$. Finally, δ is a guess of the outdegree sequence of the out-tree $B[R]$.

In Line 8 we create a flow network, and in line 9 a minimum cost maximum flow is found in polynomial time. In the next section we discuss the flow network and properties of the flow.

6.2 The flow

In this section we consider a run of Algorithm 2, and in particular we assume that the variables $R, \delta, L_1, \ldots, L_{K+1}$ have been assigned accordingly. The function CreateNetwork in our algorithm builds a flow network $F = (V(F), E(F), \mathsf{cap}, \mathsf{cost})$, where $E(F)$ is a set of directed edges and cap and cost are functions from edges to integers denoting capacities and costs of corresponding edges. As usual, the function cost extends to flow functions in a natural way, i.e., $\mathsf{cost}(f) = \sum_{e \in E(F)} f(e)\mathsf{cost}(e)$. We let $V(F) = \{s, t\} \cup \{v^I, v^O \mid v \in V(G)\} \cup \{v^C \mid v \in V \setminus R\}$, where s and t denote the source and the sink of F.

We put following edges into $E(F)$:

- (s, v^O), where $\mathsf{cap}(s, v^O) = \mathsf{out}'(v), \mathsf{cost}(s, v^O) = 0$ for every $v \in V(G)$
- (v^I, t), where $\mathsf{cap}(v^I, t) = \mathsf{in}'(v), \mathsf{cost}(v^I, t) = 0$ for every $v \in R$
- (v^I, t), where $\mathsf{cap}(v^I, t) = \mathsf{in}'(v) - 1, \mathsf{cost}(v^I, t) = 0$ for every $v \notin R$
- (v^C, t), where $\mathsf{cap}(v^C, t) = 1, \mathsf{cost}(v^C, t) = 0$ for every $v \notin R$
- (u^O, v^I), where $\mathsf{cap}(u^O, v^I) = \infty, \mathsf{cost}(u^O, v^I) = d(u, v)$ for every $(u, v) \in E(G)$
- (u^O, v^C), where $\mathsf{cap}(u^O, v^C) = \infty, \mathsf{cost}(u^O, v^C) = d(u, v)$ for every $v \in L_i, u \in R \cup L_{i+1} \cup \ldots \cup L_K, (u, v) \in E(G)$.

We will say that F has a *full flow* if it has a flow f with value $|f| = \sum_{v \in V} \mathsf{out}'(v)$. By the construction of F, then all edges leaving source are saturated, i.e., carry flow equal to their capacity. Since $\sum_{v \in V} \mathsf{out}'(v) = \sum_{v \in V} \mathsf{in}'(v)$, also all edges that enter the sink are saturated.

Essentially, the network above results from extending the standard network used to get Observation 3.3 by vertices v^C. The flow between $\{v^O \mid v \in V\}$ and $\{v^I \mid v \in V\} \cup \{v^C \mid v \in V \setminus R\}$ represents the resulting solution. In a full flow the edges leaving v^C are saturated, so a unit of flow enters every vertex v^C, which results in connecting v in the solution to a higher layer or to R. Thanks to that the solution resulting from adding the out-tree T_R to the solution extracted from f contains an outbranching.

▶ **Lemma 6.3.** *If f is a full flow of minimum cost in F then there exists a solution of I with cost $\mathsf{cost}(f) + d(T_R)$. Moreover, the solution can be extracted from f in polynomial time.*

Proof. By standard arguments, since all capacities in F are integer, we infer that there is an integral flow of minimum cost (and it can be found in polynomial time), so we assume w.l.o.g. that f is integral.

Let $b : V^2 \to \{0, 1\}$ denote a function such that $b(u, v) = [(u, v) \in T_R]$. Now we construct a solution $m : V^2 \to \mathbb{Z}_{\geq 0}$ of I.

$$m(u,v) = \begin{cases} f(u^O, v^I) + b(u,v) & \text{if } v \in R \\ f(u^O, v^I) + f(u^O, v^C) & \text{if } v \notin R. \end{cases}$$

In other words, m describes how many times edge (u, v) was used by the out-tree T_R and flow f in total. Let us verify that m is a feasible solution for I. The degree constraints are easy to verify, so we are left with showing that m contains an outbranching rooted at root. To this end it suffices to show that every vertex v is reachable from root in G_m. Clearly, this holds for vertices in R, thanks to the out-tree T_R. Pick an arbitrary vertex $v \notin R$. Then $v \in L_i$ for some $i = 1, \ldots, K$. We know that $f(v^C, t) = 1$, so there exists u such that $f(u^O, v^C) = 1$. Therefore, v is connected in G_m to a vertex from $R \cup L_{i+1} \cup \ldots L_K$. Since v in G_m has an in-neighbor either in R or in a layer with a higher index, we can conclude that there is a path in G_m from R to v. Hence m indeed contains the required outbranching.

Finally, it can be easily checked that $d(m) = \mathsf{cost}(f) + d(T_R)$, what concludes this proof. ◀

Let m be a feasible solution for I. Let R, L_i for $i = 1, \ldots, K + 1$ be sets of vertices and δ an out-tree sequence on R, as in Algorithm 2. We say that m is *compliant* with R, $L_1, \ldots, L_{K+1}$ and δ when m contains an outbranching T rooted at root such that $R_K(T) = R$, $L_i(T) = L_i$ for $i = 1, \ldots, K + 1$ and δ is equal to the outdegree sequence of $T[R]$.

▶ **Lemma 6.4.** *Assume that there exists a solution m of I that is compliant with R, $L_1, \ldots, L_{K+1}$ and δ. Then F has a full flow f such that $\mathsf{cost}(f) + d(T_R) \leq d(m)$.*

Proof. Let T be an outbranching in m which certifies that m is compliant with $R, L_1, \ldots, L_{K+1}$ and δ. Let $p : V^2 \to \{0, 1\}$ be a function such that for every $u, v \in V$ we have $p(u, v) = [(u, v) \in T]$.

We set $f(s, u) = \mathsf{cap}(s, u)$ for all edges $(s, u) \in E(F)$ and $f(u, t) = \mathsf{cap}(u, t)$ for all edges $(u, t) \in E(F)$. If $v \in V \setminus R$ then we set $f(u^O, v^C) = p(u, v)$. For all $u, v \in V(G)$ we set $f(u^O, v^I) = m(u, v) - p(u, v)$. It can be easily checked that such function f is a full flow and $\mathsf{cost}(f) = d(m) - d(T[R])$. However, since $T[R]$ is a δ-out-tree rooted at root and T_R is a cheapest such out-tree, $d(T_R) \leq d(T[R])$. It follows that $\mathsf{cost}(f) \leq d(m) - d(T_R)$, so $\mathsf{cost}(f) + d(T_R) \leq d(m)$ as required. ◀

Consider a *minimum cost* full flow f' in F that is found by Algorithm 2 for a choice of $R, L_1, \ldots, L_{K+1}, \delta$. The claim above implies that $\mathsf{cost}(f') + d(T_R) \leq d(m)$. However, notice that we do not claim that $\mathsf{cost}(f')$ is the cost of optimal completion of T_R consistent with all guesses, as the intuitions we described earlier might suggest. It could be the case that in the solution resulting from f', a vertex which was guessed to belong to L_i does not have any out-neighbor that was guessed to belong to L_{i-1}, which would mean that this vertex should be in an earlier layer. However, that is not an issue for the extraction of the global optimum solution of I, because we may get only better solutions than the optimum completion for that particular guess.

6.3 Correctness

▶ **Lemma 6.5.** *Function* Solve *returns the cost of an optimal solution of* I.

Proof. From Lemma 6.3 we infer that Solve returns the cost of a feasible solution of I. It remains to show that it returns a value that is smaller or equal to the cost of an optimal solution of I. To this end, let m be an arbitrary optimal solution of I and let T be an arbitrary outbranching rooted at root in G_m. Let $R = R_K(T)$, $L_i = L_i(T)$ for $i = 1, \ldots, K+1$ and let δ be the outdegree sequence of $T[R]$.

Let us verify that $R, L_1, \ldots, L_{K+1}$ and δ satisfy constraints (C1)–(C5). We get (C1) and (C5) by Lemma 6.2. (C2) follows from the definition of δ. For (C3), consider two cases. If $|R| > 1$, then L_{K+1} is the set of leaves in R and hence indeed for every $v \in L_{K+1}$ we have $\delta_v = 0$ and for every $v \in R \setminus (L_{K+1} \cup \{\mathsf{root}\})$ we have $\delta_v \geq 1$. When $|R| \leq 1$, we have $L_{K+1} = \emptyset$ and since $\mathsf{root} \in R$ by Lemma 6.2, $R = \{\mathsf{root}\}$. Then both sets L_{K+1} and $R \setminus (L_{K+1} \cup \{\mathsf{root}\})$ are empty, so (C3) trivially holds. Finally, (C4) follows by the definition of leaf layers.

Since $R, L_1, \ldots, L_{K+1}$ and δ satisfy constraints (C1)–(C5), then Solve reaches this particular evaluation of the variables $R, L_1, \ldots, L_{K+1}$ and δ. Then, based on Lemma 6.4, the network F has a full flow f such that $\mathsf{cost}(f) + d(T_R) \leq d(m)$, and it follows that Solve returns a value $\mathsf{best} \leq \mathsf{cost}(f) + d(T_R) \leq d(m)$, as required. ◀

Obviously, Solve can be easily adapted to return a solution of I with the cost it returns, but we have not taken this into account in Algorithm 2 for the sake of its readability.

6.4 Running time

Having a correct algorithm solving Fixed Degree Subgraph With Outbranching in polynomial space, let us analyze its complexity depending on K.

Let us denote $r = |R|$ and $c = |L_{K+1}|$. Recall that $1 \leq r \leq n$ and $0 \leq c \leq \lfloor \frac{n-r}{K} \rfloor$.

If we fix r and c, then there are $\binom{n-1}{r-1}$ guesses for R (it has to contain root) and at most $\binom{r-1}{c}$ guesses for L_{K+1}. Let us bound the number of guesses for δ. By (C2) and (C3), $\sum_{v \in R} \delta_v = r - 1$, and $\delta_v = 0$ iff $v \in L_{K+1}$ so essentially we put $r-1$ balls into $r-c$ bins that must be nonempty, which is $\binom{r-2}{c-1}$ by standard combinatorics. In the special case $c = 0$ there is one choice for δ, where $\delta_{\mathsf{root}} = 0$.

In total, there are at most $\binom{n}{r}\binom{r}{c}^2$ guesses for all R, L_{K+1}, δ simultaneously. For each of these guesses, using Lemma 6.1 function Solve calculates an optimal δ-out-tree spanning R, which takes time $\mathcal{O}(4^{n+o(n)})$. It follows that that part takes time $\mathcal{O}^*(\sum_r \binom{n}{r} 4^{r+o(r)} \sum_c \binom{r}{c}^2)$. Then, Solve guesses a partition of $V \setminus R$ into $L_1, \ldots, L_K$ in at most K^{n-r} ways. For each such guess, Solve spends polynomial time, so that part takes $\mathcal{O}^*(\sum_r \binom{n}{r} K^{n-r} \sum_c \binom{r}{c}^2)$ time. Hence the total running time can be bounded by

$$\mathcal{O}^*\left(2^{o(n)} \sum_{r=1}^{n} \sum_{c=0}^{\lfloor \frac{n-r}{K} \rfloor} \underbrace{\binom{n}{r}(K^{n-r} + 4^r)\binom{r}{c}^2}_{\xi(r,c)}\right).$$

Since there are polynomially many guesses for r and c, we can actually replace sums with maxima in the expression above and focus on the expression $\xi(r,c) = \binom{n}{r}(K^{n-r} + 4^r)\binom{r}{c}^2$.

We will heavily use the well-known bound $\binom{n}{\alpha n} < 2^{h(\alpha)n}$, where $h(\alpha) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$ is the binary entropy function (see e.g. [13]). For readability, let us denote $f(\alpha) = 2^{h(\alpha)}$ and let us point out that f is increasing on the interval $[0, \frac{1}{2}]$ and decreasing on the interval $[\frac{1}{2}, 1]$. Let us denote $\beta \coloneqq \frac{r}{n}$. We are going to distinguish two cases here.

1. $\frac{n-r}{K} \geq \frac{r}{2}$
   This inequality can be rephrased as $r \leq \frac{2}{K+2}n$, which is equivalent to $\beta \leq \frac{2}{K+2}$. We will use here a trivial bound $\binom{r}{c} \leq 2^r$. Then, $\xi(r,c) \leq f(\beta)^n((K^{1-\beta})^n + 4^{\beta n})4^{\beta n} = (f(\beta)K^{1-\beta}4^{\beta})^n + (f(\beta)4^{2\beta})^n$
2. $\frac{n-r}{K} < \frac{r}{2}$
   In that case we know that $\max_{c=0}^{\lfloor \frac{n-r}{K} \rfloor} \binom{r}{c}^2$ is attained when $c = \lfloor \frac{n-r}{K} \rfloor$ and for that particular value of c we can use the following bound.

$$\binom{r}{c}^2 = \left(\frac{r}{\lfloor \frac{n-r}{K} \rfloor} \cdot r\right)^2 = \mathcal{O}^*\left(f\left(\frac{\lfloor \frac{n-r}{K} \rfloor}{r}\right)^{2r}\right) = \mathcal{O}^*\left(f\left(\frac{\frac{n-r}{K}}{r}\right)^{2r}\right) =$$

$$= \mathcal{O}^*\left(f\left(\frac{1-\beta}{K\beta}\right)^{2\beta n}\right)$$

   In the third equality above we used fact that f is increasing on the interval $[0, \frac{1}{2}]$. To sum up, in this case,

$$\xi(r,c) = \mathcal{O}^*\left(\left(f(\beta)K^{1-\beta}f\left(\frac{1-\beta}{K\beta}\right)^{2\beta}\right)^n + \left(f(\beta)4^{\beta}f\left(\frac{1-\beta}{K\beta}\right)^{2\beta}\right)^n\right).$$

Our numerical analysis shows that it is optimal to choose $K = 4$. For that particular value of K, the first case applies if and only if $\beta \leq \frac{1}{3}$. Then,

$$\xi(r,c) = (4f(\beta))^n + (4^{2\beta}f(\beta))^n = \mathcal{O}((4f(\beta))^n) = \mathcal{O}((4f(\tfrac{1}{3}))^n) = \mathcal{O}(7.56^n).$$

Let us now investigate the second case, when $\beta > \frac{1}{3}$. For $\frac{1}{3} < \beta < \frac{1}{2}$ the summand $\left(f(\beta)4^{1-\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta}\right)^n$ dominates, and for $\beta \geq \frac{1}{2}$ the summand $\left(f(\beta)4^{\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta}\right)^n$ dominates. We have numerically verified that

$$f(\beta)4^{1-\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta} \leq 7.68 \text{ for } \beta \in (\tfrac{1}{3}, \tfrac{1}{2})$$

and

$$f(\beta)4^{\beta}f\left(\frac{1-\beta}{4\beta}\right)^{2\beta} \leq 7.871 \text{ for } \beta \in [\tfrac{1}{2}, 1].$$

Hence, we can conclude that for $K = 4$ and our algorithm runs in time $\mathcal{O}^*(7.871^{n+o(n)}) = \mathcal{O}(7.88^n)$ and in polynomial space. This concludes the proof of Theorem 1.4.

7 $(1+\epsilon)$-approximation

In this section we show theorem 1.2, i.e. we present an algorithm for Many Visits TSP which finds a $(1+\epsilon)$-approximation in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time and polynomial space.

To achieve this we consider a more general problem, namely Fixed Degree Connected Subgraph. The main idea is to round weights of edges of the given instance, so that we can use the algorithm for polynomially bounded weights from Lemma 4.12 which is an analog of theorem 1.1 for Fixed Degree Connected Subgraph.

Let us first consider the case with degrees bounded by a polynomial.

▶ **Lemma 7.1.** *For given $\epsilon > 0$ and an instance $I = (d, \mathsf{in}, \mathsf{out})$ of* Fixed Degree Connected Subgraph *such that $\mathsf{in}(v), \mathsf{out}(v) \leq \mathcal{O}(n^2)$ for every vertex v there exists an algorithm finding a $(1+\epsilon)$-approximate solution in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time and polynomial space.*

Proof. Let us denote the optimal solution for I by OPT. First, our algorithm guesses the most expensive edge used by OPT. Let us denote its cost by E, in particular

$$E \leq d(\mathrm{OPT}). \tag{1}$$

Let us denote by C the universal constant such that $\mathsf{in}(v), \mathsf{out}(v) \leq Cn^2$ for every vertex v and let us round d in the following way

$$d'(u,v) := \begin{cases} \left\lceil \frac{Cn^3}{\epsilon E} d(u,v) \right\rceil & \text{if } d(u,v) \leq E \\ \infty & \text{if } d(u,v) > E \end{cases} \tag{2}$$

If $d'(u,v)$ is finite then it is bounded by $\left\lceil \frac{Cn^3}{\epsilon E} E \right\rceil = \left\lceil \frac{Cn^3}{\epsilon} \right\rceil$. Our algorithm simply returns the optimal solution for instance $I' = (d', \mathsf{in}, \mathsf{out})$ which can be found in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time using the algorithm from Lemma 4.12 with $D = \left\lceil \frac{Cn^3}{\epsilon} \right\rceil$. Let us denote this solution by ALG. Now we only need to prove that ALG is a $(1+\epsilon)$-approximation for the original instance I. We know that ALG is an optimal solution for I', in particular

$$d'(\mathrm{ALG}) \leq d'(\mathrm{OPT}). \tag{3}$$

For every v we have $\mathsf{out}(v) \leq Cn^2$, so

$$\sum_{(u,v) \in V^2} \mathrm{OPT}(u,v) = \sum_{u \in V} \mathsf{out}(u) \leq n \cdot Cn^2. \tag{4}$$

The following chain of inequalities finishes the proof.

$$d(\mathrm{ALG}) \overset{(2)}{\leq} \frac{\epsilon E}{Cn^3} d'(\mathrm{ALG}) \overset{(3)}{\leq} \frac{\epsilon E}{Cn^3} d'(\mathrm{OPT}) \overset{(2)}{\leq} \frac{\epsilon E}{Cn^3} \sum_{(u,v) \in V^2} \mathrm{OPT}(u,v) \left(\frac{d(u,v) Cn^3}{\epsilon E} + 1 \right) =$$

$$= d(\mathrm{OPT}) + \frac{\epsilon E}{Cn^3} \sum_{(u,v) \in V^2} \mathrm{OPT}(u,v) \overset{(4)}{\leq} d(\mathrm{OPT}) + \frac{\epsilon E}{Cn^3} Cn^3 \overset{(1)}{\leq} (1+\epsilon) d(\mathrm{OPT}) \qquad \blacktriangleleft$$

Now we can generalize the algorithm from Lemma 7.1 by using it as a black box for the general case.

▶ **Lemma 7.2.** *For a given $\epsilon > 0$ and an instance $I = (d, \mathsf{in}, \mathsf{out})$ of* Fixed Degree Connected Subgraph *there exists an algorithm finding a $(1+\epsilon)$-approximate solution in $\mathcal{O}^*\left(\frac{2^n}{\epsilon}\right)$ time and polynomial space.*

Proof. First let us use the algorithm from Theorem 3.4 which outputs an instance $I' = (d, \mathsf{in}', \mathsf{out}')$ of Fixed Degree Connected Subgraph and a function $f : V^2 \to \mathbb{Z}_{\geq 0}$. Let us denote the optimal solution for I' by OPT'. By Theorem 3.4 the optimal solution for I equals $\mathrm{OPT}' + f$. Moreover, we know that $\mathsf{in}', \mathsf{out}'(v) \leq \mathcal{O}(n^2)$ for every vertex v. In particular we can use algorithm from Lemma 7.1 to get a solution ALG' for instance I' such that $d(\mathrm{ALG}') \leq (1+\epsilon) d(\mathrm{OPT}')$. Our algorithm simply returns solution $\mathrm{ALG}' + f$, which is a solution for I because ALG is connected and f increases degrees exactly by difference between I and I'. To prove $\mathrm{ALG}' + f$ is $(1+\epsilon)$-approximation we just need to observe that

$$d(\mathrm{ALG}' + f) \leq (1+\epsilon) d(\mathrm{OPT}') + d(f) \leq (1+\epsilon) d(\mathrm{OPT}' + f). \qquad \blacktriangleleft$$

Fixed Degree Connected Subgraph is a generalization of Many Visits TSP so the algorithm from Lemma 7.2 proves Theorem 1.2.

8 Further Research

Since TSP is solvable in time $\mathcal{O}^*(2^n)$ and exponential space [1, 16] and time $\mathcal{O}(4^{n+o(n)})$ and polynomial space [15], the main remaining question is whether these bounds can be achieved for Many Visits TSP avoiding in the running time bound the linear dependence on maximum distance D. Another interesting goal is a deterministic version of Theorem 4.1.

References

1 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, January 1962. doi:10.1145/321105.321111.

2 André Berger, László Kozma, Matthias Mnich, and Roland Vincze. Time- and space-optimal algorithms for the many-visits TSP. *CoRR*, abs/1804.06361, 2018. arXiv:1804.06361.

3 André Berger, László Kozma, Matthias Mnich, and Roland Vincze. A time- and space-optimal algorithm for the many-visits TSP. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1770–1782. SIAM, 2019. doi:10.1137/1.9781611975482.106.

4 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed Hamiltonicity and out-branchings via generalized Laplacians. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 91:1–91:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.91.

5 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Fast witness extraction using a decision oracle. In *ESA*, volume 8737 of *Lecture Notes in Computer Science*, pages 149–160. Springer, 2014.

6 Nadia Brauner, Yves Crama, Alexander Grigoriev, and Joris Van de Klundert. A framework for the complexity of high-multiplicity scheduling problems. *J. Comb. Optim.*, 9:313–323, May 2005. doi:10.1007/s10878-005-1414-7.

7 Stavros S. Cosmadakis and Christos H. Papadimitriou. The traveling salesman problem with many visits to few cities. *SIAM J. Comput.*, 13(1):99–108, 1984.

8 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3), May 2016. doi:10.1145/2925416.

9 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3), March 2018. doi:10.1145/3148227.

10 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159. IEEE Computer Society, 2011.

11 R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7:193–195, 1978.

12 Hamilton Emmons and Kamlesh Mathur. Lot sizing in a no-wait flow shop. *Operations Research Letters - ORL*, 17:159–164, May 1995. doi:10.1016/0167-6377(95)00008-8.

13 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

14 Alexander Grigoriev and Joris Van de Klundert. On the high multiplicity traveling salesman problem. *Discrete Optimization*, 3:50–62, March 2006. doi:10.1016/j.disopt.2005.11.002.

15 Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, June 1987. doi:10.1137/0216034.

16 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, page 71.201–71.204, New York, NY, USA, 1961. Association for Computing Machinery. doi:10.1145/800029.808532.

17 Dorit S. Hochbaum and Ron Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Oper. Res.*, 39:648–653, 1991.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

20 László Kozma and Tobias Mömke. Maximum scatter TSP in doubling metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, page 143–153, USA, 2017. Society for Industrial and Applied Mathematics.

21 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. In *Algorithms – ESA 2010*, volume 64, pages 549–560, September 2010. doi:10.1007/978-3-642-15775-2_47.

22 László Lovász. On determinants, matchings and random algorithms. In *FCT*, volume 79, pages 565–574, January 1979.

23 James B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.*, 41(2):338–350, 1993. doi:10.1287/opre.41.2.338.

24 Harilaos N. Psaraftis. A dynamic programming approach for sequencing groups of identical jobs. *Operations Research*, 28(6):1347–1359, 1980.

25 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

26 W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, s1-22(2):107–111, April 1947. doi:10.1112/jlms/s1-22.2.107.

27 Jack A. A. van der Veen and Shuzhong Zhang. Low-complexity algorithms for sequencing jobs with a fixed number of job-classes. *Comput. Oper. Res.*, 23(11):1059–1067, November 1996. doi:10.1016/0305-0548(96)00016-0.

28 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. International Symposium on Symbolic and Algebraic Computation*, volume 72 of *LNCS*, pages 216–226, 1979.