The Asymmetric Travelling Salesman Problem In Sparse Digraphs

Łukasz Kowalik
Institute of Informatics, University of Warsaw, Poland
kowalik@mimuw.edu.pl

Konrad Majewski
Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland
km371194@students.mimuw.edu.pl

Abstract

Asymmetric Travelling Salesman Problem (ATSP) and its special case Directed Hamiltonicity are among the most fundamental problems in computer science. The dynamic programming algorithm running in time $\mathcal{O}^*(2^n)$ developed almost 60 years ago by Bellman, Held and Karp, is still the state of the art for both of these problems.

In this work we focus on *sparse* digraphs.

First, we recall known approaches for Undirected Hamiltonicity and TSP in sparse graphs and we analyse their consequences for Directed Hamiltonicity and ATSP in sparse digraphs, either by adapting the algorithm, or by using reductions. In this way, we get a number of running time upper bounds for a few classes of sparse digraphs, including $\mathcal{O}^*(2^{n/3})$ for digraphs with both out- and indegree bounded by 2, and $\mathcal{O}^*(3^{n/2})$ for digraphs with outdegree bounded by 3.

Our main results are focused on digraphs of bounded *average* outdegree d. The baseline for ATSP here is a simple enumeration of cycle covers which can be done in time bounded by $\mathcal{O}^*(\mu(d)^n)$ for a function $\mu(d) \le (\lceil d \rceil !)^{1/\lceil d \rceil}$. One can also observe that Directed Hamiltonicity can be solved in randomized time $\mathcal{O}^*((2-2^{-d})^n)$ and polynomial space, by adapting a recent result of Björklund [ISAAC 2018] stated originally for Undirected Hamiltonicity in sparse bipartite graphs. We present two new deterministic algorithms for ATSP: the first running in time $\mathcal{O}(2^{0.441(d-1)n})$ and polynomial space, and the second in exponential space with running time of $\mathcal{O}^*(\tau(d)^{n/2})$ for a function $\tau(d) \le d$.

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

Keywords and phrases asymmetric traveling salesman problem, Hamiltonian cycle, sparse graphs, exponential algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.23

Related Version A full version of the paper is available at [27], `https://arxiv.org/abs/2007.12120`.

Funding *Łukasz Kowalik*: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).

Acknowledgements The authors thank the reviewers for careful reading and useful comments.

1 Introduction

In the Directed Hamiltonicity problem, given a directed graph (digraph) G one has to decide if G has a Hamiltonian cycle, i.e., a simple cycle that visits all vertices. In its weighted version, called ATSP, we additionally have integer weights on edges $w : E \to \mathbb{Z}$, and the goal is to find a minimum weight Hamiltonian cycle in G.

The ATSP problem has a dynamic programming algorithm running in time and space $\mathcal{O}^*(2^n)$ due to Bellman [2] and Held and Karp [23]. Gurevich and Shelah [22] obtained the best known *polynomial space* algorithm, running in time $\mathcal{O}(4^n n^{\log n})$. It is a major open problem whether there is an algorithm in time $\mathcal{O}^*((2-\varepsilon)^n)$ for an $\varepsilon > 0$, even for the unweighted case of Directed Hamiltonicity. However, there has been a significant progress in answering this question in variants of Directed Hamiltonicity. Namely, Björklund and Husfeldt [6] showed that the parity of the number of Hamiltonian cycles in a digraph can be determined in time $O(1.619^n)$ and Cygan, Kratsch and Nederlof [13] solved the bipartite case of Directed Hamiltonicity in time $\mathcal{O}(1.888^n)$, which was later improved to $\mathcal{O}^*(3^{n/2}) = O(1.74^n)$ by Björklund, Kaski and Koutis [9].

Table 1 Running times (with polynomial factors omitted) of algorithms for *undirected* graphs. Rows marked with 💣 denote exponential space algorithms, rows marked with ⚅ denote Monte Carlo algorithms.

Graph class	Undirected Hamiltonicity			Travelling Salesman Problem		
general	1.66^n	⚅	[3]	2^n $4^n n^{\log n}$	💣	[2, 23] [22]
bipartite	1.42^n	⚅	[3]	2^n 4^n	💣	[2, 23] [29]
$\Delta = 3$	1.16^n 1.24^n	💣 ⚅	[13] [33]	1.22^n 1.24^n	💣	[11] [33]
$\Delta = 4$	1.51^n 1.59^n	💣 ⚅ ⚅	[13]+[17] [8]	1.63^n 1.70^n	💣	[11]+[17] [34]
$\Delta = 5$	1.63^n	⚅	[8]	1.88^n 2.35^n	💣	[11]+[17] [35]
any Δ				$(2-\varepsilon'_\Delta)^n$	💣	[7]
avgdeg $\le d$	1.12^{dn}	💣 ⚅	[13]+[25]	1.14^{dn} $2^{(1-\varepsilon_d)n}$	💣 💣	[11]+[25] [15]
bipartite avgdeg $\le d$	$(2-2^{1-d})^{n/2}$	⚅	[4]			
pathwidth	$3.42^{\mathtt{pw}}$	💣 ⚅	[13]	$4.28^{\mathtt{pw}}$	💣	[11]
treewidth	$4^{\mathtt{tw}}$	💣 ⚅	[14]	$9.56^{\mathtt{tw}}$	💣	[11]

Undirected graphs. Even more is known in the undirected setting, where the problems are called Undirected Hamiltonicity and TSP. Björklund [3] shows that Undirected Hamiltonicity can be solved in time $\mathcal{O}(1.66^n)$ in general and $\mathcal{O}^*(2^{n/2}) = O(1.42^n)$ in the bipartite case. Very recently, Nederlof [28] showed that the bipartite case of TSP admits an algorithm in time $\mathcal{O}(1.9999^n)$, assuming that square matrices can be multiplied in time $O(n^{2+o(1)})$. Finally, there is a number of results for Undirected Hamiltonicity and TSP restricted to graphs that are somewhat sparse. An early example is an algorithm of Eppstein [16] for TSP in graphs of maximum degree 3, running in time $\mathcal{O}^*(2^{n/3}) = O(1.26^n)$. This result has been later improved and generalized to larger values of maximum degree, we refer the reader to Table 1 for details (Δ denotes the maximum degree). Perhaps the most general measure of graph sparsity is the average degree d. Cygan and Pilipczuk [15] showed that whenever d is bounded, the 2^n barrier for TSP can be broken, although only slightly. More precisely, they proved the bound $\mathcal{O}^*(2^{(1-\varepsilon_d)n})$, where $\varepsilon_d = 1/(2^{2d+1} \cdot 20d \cdot e^{e^{20d}})$. We note that although their result was stated for undirected graphs, the same reasoning can be made for digraphs of average *total degree* (sum of indegree and outdegree). For small values of d, more significant improvements are possible. Namely, by combining the algorithms for Undirected Hamiltonicity and TSP parameterized by pathwidth [11, 13] with a bound on pathwidth of sparse graphs [25] we get the upper bound of $\mathcal{O}(1.12^{dn})$ and $\mathcal{O}(1.14^{dn})$, respectively. For Undirected Hamiltonicity, if the input graph is additionally bipartite, Björklund [4] shows the $\mathcal{O}^*((2-2^{1-d})^{n/2})$ upper bound.

Table 2 Running times (with polynomial factors omitted) of the algorithms for *directed* graphs. We preserve the notation from Table 1. By Δ^+ we denote maximum outdegree and Δ denotes maximum total degree. Treewidth refers to the underlying undirected graph.

Graph class	Directed Hamiltonicity			Asymmetric Travelling Salesman Problem		
general	2^n		[1, 24, 26]	2^n $4^n n^{\log n}$	💣	[2, 23] [22]
bipartite	1.74^n	⚅	[9]	2^n 4^n	💣	[2, 23] [29]
$(2,2)$-graphs	1.26^n		(Corollary 2.6)	1.26^n		(Corollary 2.6)
$\Delta^+ = 3$	1.74^n	💣	(Corollary 2.8)	1.74^n	💣	(Corollary 2.8)
$\Delta = 3$	1.13^n		(Corollary 2.7)	1.13^n		(Corollary 2.7)
any Δ	$(2-2^{-\Delta/2})^n$	⚅	(Theorem 2.10)	$(2-\varepsilon'_\Delta)^n$	💣	[7]
average outdeg $\leq d$	$\mu(d)^n$ $2^{0.441(d-1)n}$ $\sqrt{\tau(d)}^n$ $(2-2^{-d})^n$ $2^{(1-\Omega(1/d))n}$	 💣 ⚅ ⚅	(Corollary 2.4) (Theorem 1.1) (Theorem 1.2) (Theorem 2.10) [5]	$\mu(d)^n$ $2^{0.441(d-1)n}$ $\sqrt{\tau(d)}^n$ $2^{(1-\varepsilon_{2d})n}$	 💣 💣	(Corollary 2.4) (Theorem 1.1) (Theorem 1.2) [15]
treewidth	$6^{\mathtt{tw}}$	💣 ⚅	[14]			

Directed sparse graphs: hidden results. The goal of this paper is to investigate Directed Hamiltonicity and ATSP in *sparse directed* graphs. Quite surprisingly, not much results in this topic are stated explicitly. In fact, we were able to find just a few references of this kind: Björklund, Husfeldt, Kaski and Koivisto [7] describe an algorithm for digraphs with total degree bounded by D that works in time $\mathcal{O}^*((2-\varepsilon'_D)^n)$, for $\varepsilon'_D = 2-(2^{D+1}-2D-2)^{1/(D+1)}$. Second, Cygan et al. [14] describe an algorithm for Directed Hamiltonicity running in time $6^t n^{O(1)}$, where t is the treewidth of the input graph. Finally, Björklund and Williams [10] show a deterministic algorithm which *counts* Hamiltonian cycles in directed graphs of average degree d in time $2^{n-\Omega(n/d)}$ and exponential space. Very recently, Björklund [5], using a different approach, obtained the same running time for the *decision* Directed Hamiltonicity problem, but lowering the space to polynomial, at the cost of using randomization. The authors of these two works have not put an effort to optimize the constants hidden in the Ω notation. By following the analysis in each of these papers as-is, we get the saving term in the exponent at least $n/(111d)$ (for a faster, randomized algorithm) and $n/(500d)$, respectively.

However, one cannot say that nothing more is known, because many results for undirected graphs imply some running time bounds in the directed setting. We devote the first part of this work to investigating such implications. In some cases, the implications are immediate.

For example, Gebauer [20, 21] shows an algorithm running in time $\mathcal{O}^*(3^{n/2}) = \mathcal{O}^*(1.74^n)$ that solves TSP in graphs of maximum degree 4. It uses the meet-in-the-middle approach and can be sketched as follows: guess two opposite vertices of the solution cycle, generate a family of paths of length $n/2$ from each of them (of size at most $3^{n/2}$) and store one of the families in a dictionary to enumerate all complementary pairs of paths in time $\mathcal{O}^*(3^{n/2})$. This algorithm, without a change, can be used for ATSP in digraphs of maximum outdegree 3, with the same running time bound (see Theorem 2.8).

The other implications that we found rely on a simple reduction from ATSP to a variant of TSP in *bipartite undirected* graphs (see Lemma 2.1): replace each vertex v of the input digraph G by two vertices v^{out}, v^{in} joined by an edge of weight 0, and for each edge $(u,v) \in E(G)$ create an edge $u^{\mathsf{out}}v^{\mathsf{in}}$ of the same weight. Then find a lightest Hamiltonian cycle that contains the matching $M = \{v^{\mathsf{out}}v^{\mathsf{in}} \mid v \in V(G)\}$. By applying this reduction to a digraph with both outdegrees and indegrees bounded by 2, which we call a $(2,2)$-graph, and using Eppstein's algorithm [16] we get the running time of $\mathcal{O}^*(2^{n/3}) = \mathcal{O}^*(1.26^n)$, see Corollary 2.6. Another consequence is an algorithm running in time $\mathcal{O}^*(2^{n/6})$ for digraphs of maximum total degree 3, see Corollary 2.7. These two simple classes of digraphs were studied by Plesník [30], who showed that Directed Hamiltonicity remains NP-complete when restricted to them.

We can also apply the reduction to an arbitrary digraph of average outdegree d. A naive approach would be then to enumerate all perfect matchings in the bipartite graph induced by edges $\{u^{\mathsf{out}}v^{\mathsf{in}} \mid (u,v) \in E(G)\}$. Indeed, each such matching corresponds to a cycle cover in the input graph, so we basically enumerate cycle covers and filter-out the disconnected ones. Thanks to Bregman-Minc inequality [12] which bounds the permament in sparse matrices the resulting algorithm has running time $\mathcal{O}^*(\mu(d)^n)$, where

$$\mu(d) = (\lfloor d \rfloor !)^{\frac{\lfloor d \rfloor + 1 - d}{\lfloor d \rfloor}} (\lceil d \rceil !)^{\frac{d - \lfloor d \rfloor}{\lceil d \rceil}} \leq (\lceil d \rceil !)^{1/\lceil d \rceil}.$$

See Corollary 2.4 for details.

Yet another upper bound for digraphs of average outdegree d is obtained by using the reduction described above and next applying Björklund's algorithm for sparse bipartite graphs [4] with a slight modification to force the matching M in the Hamiltonian cycle (see Theorem 2.10). The resulting algorithm has running time $\mathcal{O}^*((2 - 2^{-d})^n)$.

Directed sparse graphs: main results. The simple consequences that we describe above are complemented by two more technical results.

The first algorithm runs in polynomial space and realizes the following idea. Assume $d < 3$. Then many of the vertices of the input graph have outdegree at most 2, and we can just branch on vertices of outdegree at least 3, and solve the resulting $(2,2)$-graph using the fast $\mathcal{O}^*(2^{n/3})$-time algorithm mentioned before. This idea can be boosted a bit in the case when the initial branching is too costly, i.e., there are many vertices of high outdegree: then we observe that in such an unbalanced graph one can apply the simple cycle cover enumeration which then runs faster than in graphs of the same density but with balanced outdegrees. After a technical analysis of the running time we get the following theorem.

▶ **Theorem 1.1.** ATSP *restricted to digraphs of average outdegree at most d can be solved in time $\mathcal{O}^*(2^{\alpha(d-1)n})$ and polynomial space, where $\alpha = \frac{7}{12} - \frac{1}{12(\log_2 3 - 1)} < 0.44088$.*

The second algorithm generalizes Gebauer's meet-in-the-middle approach to digraphs of average outdegree d. (We note that it uses exponential space.)

▶ **Theorem 1.2.** ATSP *restricted to digraphs of average outdegree at most d can be solved in time $\mathcal{O}^*(\tau(d)^{n/2})$ and the same space, where*

$$\tau(d) = \lfloor d\rfloor^{\lfloor d\rfloor+1-d}(\lfloor d\rfloor+1)^{d-\lfloor d\rfloor} \le d$$

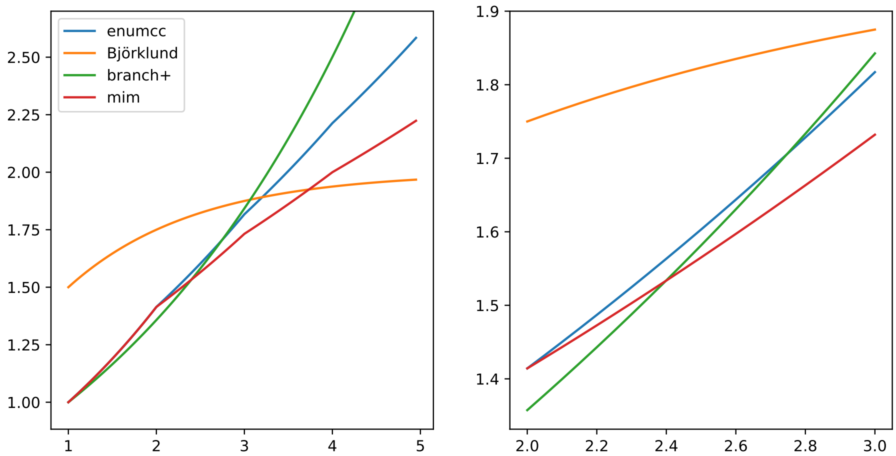

Figure 1 Comparison of the running times of algorithms for solving ATSP (enumcc, branch+, mim) and Directed Hamiltonicity (Björklund) in sparse digraphs. Horizontal axis: average degree d, vertical axis: base b from the running time bound of the form $\mathcal{O}^*(b^n)$.

Which algorithm is the best? Figure 1 compares four algorithms for solving ATSP and Directed Hamiltonicity in digraphs of average outdegree d described above:

- enumcc: enumerating cycle covers (Corollary 2.4),
- Björklund: adaptation of Björklund's bipartite graphs algorithm (Theorem 2.10),
- branch+: branching boosted by enumerating cycle covers (Theorem 1.1).
- mim: meet in the middle (Theorem 1.2),

The choice of the best (in terms of the asymptotic worst-case running time) algorithm depends on d, on whether we can afford exponential space, and on whether we solve ATSP or just Directed Hamiltonicity. We can conclude the following.

- **ATSP in polynomial space:** for $d < 2.746$ use branch+, for $d \in [2.746, 8.627]$ use enumcc, and for $d > 8.627$ use the general algorithm of Gurevich and Shelah [22].
- **ATSP in exponential space:** for $d < 2.398$ use branch+, for $d \in [2.398, 3.999]$ use mim, and for $d > 3.999$ use the algorithm of Cygan and Pilipczuk [15].
- **Directed Hamiltonicity in polynomial space:** for $d < 2.746$ use branch+, for $d \in [2.746, 3.203]$ use enumcc, for $d > 3.203$ use Björklund, and for sufficiently large d use the algorithm of Björklund [5].
- **Directed Hamiltonicity in exponential space:** for $d < 2.398$ use branch+, for $d \in [2.398, 3.734]$ use mim, for $d > 3.734$ use Björklund, and for sufficiently large d use the algorithm of Björklund and Williams [10].

2 Reductions from undirected graphs

The objective of this section is to recall two reductions from the ATSP to the (forced) TSP. Then, we will discuss existing methods of solving Undirected Hamiltonicity and TSP, and present their implications for corresponding problems in directed graphs. The summary of this section is presented in Tables 1 and 2.

2.1 General reductions

We recall that in the Forced Travelling Salesman Problem [16, 31, 33, 34], we are given an undirected graph G, a weight function $w : E(G) \to \mathbb{Z}$, and a subset $F \subseteq E(G)$. We say that a Hamiltonian cycle H is *admissible*, if $F \subseteq H$. The goal is to find an admissible Hamiltonian cycle of the minimum total weight of the edges (or, report that there is no such cycle). Moreover, we define the Bipartite Forced Matching TSP (BFM-TSP) as a special case of the Forced TSP, where graph G is bipartite, and the edges of F form a perfect matching in G.

The following lemma provides the relationship between the BFM-TSP and the ATSP.

▶ **Lemma 2.1.** *For every instance (G, w) of* ATSP*, where G is a digraph on n vertices, there is an equivalent instance $(\widehat{G}, \widehat{w}, M)$ of* BFM-TSP *such that $\widehat{G}$ is a graph on $2n$ vertices.*

Moreover, if both outdegrees and indegrees of G are bounded by D, then $\widehat{G}$ has maximum degree D. Similarly, if G has average outdegree d, then $\widehat{G}$ has average degree $d + 1$.

Proof. The proof is based on the folklore reduction from ATSP to TSP. Let (G, w) be an instance of ATSP. Let $V^{\mathsf{out}} = \{v^{\mathsf{out}} \mid v \in V(G)\}$ and $V^{\mathsf{in}} = \{v^{\mathsf{in}} \mid v \in V(G)\}$. We define $\widehat{G}$ as a bipartite graph on the vertex set $V(\widehat{G}) = V^{\mathsf{out}} \cup V^{\mathsf{in}}$ with edges $E(\widehat{G}) = \{u^{\mathsf{out}}v^{\mathsf{in}} \mid (u, v) \in E(G)\} \cup M$, where M is the perfect matching $M = \{v^{\mathsf{in}}v^{\mathsf{out}} \mid v \in V(G)\}$. The edges of $E(\widehat{G}) \setminus M$ inherit the weight from G, i.e. for $(u, v) \in E(G)$ we set $\widehat{w}(u^{\mathsf{out}}v^{\mathsf{in}}) = w(uv)$. Edges of M have weight 0. It is easy to see that the instance has the desired properties (deferred to the full version due to space constraints). ◀

Lemma 2.1 implies, in particular, that if there is an algorithm for BFM-TSP running in time $\mathcal{O}^*(f(n))$, then there is an algorithm for ATSP running in time $\mathcal{O}^*(f(2n))$.

2.2 Enumerating cycle covers

Let $(\widehat{G}, \widehat{w}, M)$ be an instance of BFM-TSP, and let $\mathcal{M}$ be a family of all perfect matchings in $\widehat{G} - M$. We observe that every cycle cover in $\widehat{G}$ which contains all edges of M is of the form $M \cup M'$, where $M' \in \mathcal{M}$. Hence, our goal is to find a matching $M' \in \mathcal{M}$ such that $M \cup M'$ is a Hamiltonian cycle in $\widehat{G}$, and the weight of M' is minimum possible. One way to do it is to list all the perfect matchings $M' \in \mathcal{M}$, and choose the best one among these which form with M a Hamiltonian cycle in $\widehat{G}$. We will investigate the complexity of such an approach in sparse graphs.

It is known that all perfect matchings in bipartite graph $\widehat{G}$ can be listed in time $|\mathcal{M}|n^{\mathcal{O}(1)}$ and polynomial space [18]. Hence, it is enough to provide a bound on the size of $\mathcal{M}$ in sparse graphs. We start with recalling a classic result of Bregman.

▶ **Theorem 2.2** (Bregman-Minc inequality [12, 32])**.** *Let A be an $n \times n$ binary matrix, and let r_i denote the number of ones in the i-th row. Then*

$$\operatorname{per} A \le \prod_{i=1}^{n} (r_i!)^{1/r_i}.$$

▶ **Corollary 2.3.** ATSP *restricted to digraphs of outdegree bounded by* D *can be solved in time* $(D!)^{n/D} n^{\mathcal{O}(1)}$ *and polynomial space.*

Proof. Given an instance (G, w) of ATSP, we use Lemma 2.1 to obtain an equivalent instance $(\widehat{G}, \widehat{w}, M)$ of BFM-TSP. Then, $H := \widehat{G} - M$ is a bipartite graph on $V^{\mathsf{out}} \cup V^{\mathsf{in}}$, and all vertices of V^{out} in H have degree at most D. Since the number of perfect matchings coincides with the permanent of the adjacency matrix, and vertex degrees correspond to the number of ones in the corresponding rows, by Theorem 2.2, there are at most $(D!)^{n/D}$ perfect matchings in H. Hence, according to our initial observation, the instance $(\widehat{G}, \widehat{w}, M)$ can be solved in time $(D!)^{n/D} n^{\mathcal{O}(1)}$. ◀

To the best of our knowledge, Corollary 2.3 provides the fastest polynomial space algorithm for $D \in \{3, 4, \ldots, 8\}$. The Bregman-Minc inequality is also useful for digraphs with bounded average outdegree.

▶ **Corollary 2.4 (♠[1]).** ATSP *restricted to digraphs of average outdegree* d *can be solved in time* $\mu(d)^n n^{\mathcal{O}(1)}$ *and polynomial space, where*

$$\mu(d) = (\lfloor d \rfloor!)^{\frac{\lfloor d \rfloor + 1 - d}{\lfloor d \rfloor}} (\lceil d \rceil!)^{\frac{d - \lfloor d \rfloor}{\lceil d \rceil}}$$

In particular, for integral values of d*, the running time is bounded by* $(d!)^{n/d} n^{\mathcal{O}(1)}$*.*

2.3 Branching algorithms

One of the most common techniques which is used for solving NP-hard problems in sparse graphs is branching (bounded search trees). It is based on optimizing exhaustive search algorithms by bounding the size of the recursion tree. In case of TSP, the first result of this kind is due to Eppstein [16], and can be stated as follows.

▶ **Theorem 2.5 ([16]).** Forced TSP *restricted to subcubic graphs can be solved in time* $2^{(n-|F|)/3} n^{\mathcal{O}(1)}$ *and polynomial space.*

▶ **Corollary 2.6.** ATSP *restricted to digraphs with all out- and indegrees at most* 2 *can be solved in time* $\mathcal{O}^*(2^{n/3})$ *and polynomial space.*

Proof. Let (G, w) be an instance of ATSP, where G is a digraph with all out- and indegrees at most 2. We apply Lemma 2.1 to obtain an equivalent instance $(\widehat{G}, \widehat{w}, M)$ of BFM-TSP. We know that $\widehat{G}$ has $2n$ vertices, and is subcubic. Moreover, $(\widehat{G}, \widehat{w}, M)$ is an instance of Forced TSP with $|M| = n$ forced edges. Hence, we can use Theorem 2.5 to solve it in time $\mathcal{O}^*(2^{(2n-n)/3}) = \mathcal{O}^*(2^{n/3})$. ◀

By combining a simple reduction implicit in a paper of Plesník [30] with Corollary 2.6 we obtain the following.

▶ **Corollary 2.7 (♠).** ATSP *restricted to digraphs of maximum total degree* 3 *can be solved in time* $\mathcal{O}^*(2^{n/6})$ *and polynomial space.*

[1] Proofs marked by ♠ are omitted due to space constraints and can be found in the full version of the paper [27].

2.4 Meet in the middle technique

Gebauer [20] shows an algorithm for undirected graphs of maximum degree 4 by using so-called meet in the middle technique, which can be easily applied in digraphs with outdegrees bounded by D to get the following result.

▶ **Theorem 2.8** ([20]). ATSP *restricted to digraphs with outdegrees bounded by* D *can be solved in time* $\mathcal{O}^*(D^{n/2})$ *and exponential space.*

2.5 Algebraic methods

Björklund [4] shows the following result.

▶ **Theorem 2.9** ([4]). *There is a Monte Carlo algorithm which solves* Undirected Hamiltonicity *restricted to bipartite graphs of average degree at most* d *in time* $\mathcal{O}^*((2-2^{1-d})^{n/2})$ *and polynomial space.*

It turns out that the proof of Theorem 2.9 can be modified to get the following Theorem. The idea is to use the reduction of Lemma 2.1 to get a sparse bipartite graph and modify the construction of Theorem 2.9 so that a relevant forced matching is a part of the resulting Hamiltonian cycle.

▶ **Theorem 2.10.** *There is a Monte Carlo algorithm which solves* Directed Hamiltonicity *restricted to digraphs of average outdegree at most* d *in time* $\mathcal{O}^*((2-2^{-d})^n)$ *and polynomial space.*

Proof. We assume that the reader is familiar with the proof of Theorem 2.9. We apply Lemma 2.1 and we get a bipartite undirected graph $\widehat{G}=(I\cup J,\widehat{E})$ and a perfect matching $F\subseteq\widehat{E}$. Recall that $\widehat{G}$ has $2n$ vertices and average degree at most $d+1$. The goal is to decide whether $\widehat{G}$ has a Hamiltonian cycle H that contains F.

Similarly as in [4] we define a polynomial matrix M with rows indexed by the vertices of I, and columns indexed by the vertices of J, as follows.

$$M(a,x,z)_{i,j}=\begin{cases}\sum_{k\in I\setminus\{i\}} z_{i,j}z_{j,k}(a_{j,k}+x_k) & \text{when } ij\in F,\\ z_{i,j}z_{j,k}(a_{j,k}+x_k) & \text{when } ij\notin F,\text{ but } jk\in F.\end{cases}$$

These polynomials have three types of variables: x_i for every $i\in I$, $a_{j,i}$ for every edge $ji\in\widehat{E}$, $j\in J$, $i\in I$. The third type of variable is somewhat special. Pick a fixed edge $e^*=i^*j^*\in F$. For every edge $ij\in\widehat{E}\setminus\{e^*\}$ there is one variable with two names $z_{i,j}$ and $z_{j,i}$; there are also two different variables z_{i^*,j^*} and z_{j^*,i^*}. Then we define a polynomial over a large enough field of characteristic two:

$$\phi=\sum_{x\in\{0,1\}^{n/2}}\det(M(a,x,z))$$

Now we should prove that thanks to cancellation in a field of characteristic two, $\phi=\sum_{H\in\mathcal{H}}\prod_{ij\in H} z_{i,j}$, where $\mathcal{H}$ is the set of all Hamiltonian cycles in $\widehat{G}$ which contain F. Björklund (Lemma 3 in [4]) shows this equality for the original polynomial using three observations: 1) after cancellation, the surviving terms do not contain a-variables, 2) each surviving term corresponds to a unique cycle cover in the graph, and 3) terms corresponding to non-Hamiltonian cycle covers pair-up and cancel-out, because if we reverse the lexicographically first cycle that does not contain e^*, then we get exactly the same term (and if we reverse a Hamiltonian cycle we get a different term, because of the asymmetry in defining z variables). The arguments used in [4] for proving 1)-3) still hold for the new polynomial, essentially for the same reasons.

The second ingredient of Björklund's construction is an upper bound on probability that none of the columns of $M(a, x, z)$ is identically zero, where $x \in \{0,1\}^{n/2}$ is a fixed assignment, z is the vector of all $z_{i,j}$ variables, and $a \in \{0,1\}^{n/2}$ is a *random* assignment. The calculation relies on the observation that if for a vertex $j \in J$ we have $a_{j,i} + x_i \equiv 0 \pmod 2$ for all $ij \in \widehat{E}$, then the column of j is identically zero. Note that this observation still holds for our new design. It follows that the probability bounds derived in [4] apply also in our case.

The third ingredient is efficient identification of assignments $x \in \{0,1\}^{n/2}$, for which $\det(M(a, x, z))$ is non-zero (for fixed, random, values of a). This is done by creating a Boolean variable w_v corresponding to every variable x_v and building a CNF formula such that its satisfying assignments correspond to a superset of all assignments of x_v variables that result in non-zero $\det(M(a, x, z))$. Again, the fact that the resulting formula is in CNF follows from the fact that the j-th column is non-zero if for some $i \in I$ we have $a_{j,i} + x_i \equiv 1 \pmod 2$, which is also true in our design. Finally, Björklund [4] shows how to enumarate all satisfying assignments of the CNF formula efficiently, what is not altered in any way by our changes in the design of polynomial ϕ. ◀

3 Polynomial space algorithm

This section is devoted to the proof of Theorem 1.1. We begin with introducing some additional notions, then we provide a branching algorithm which will be later used as a subroutine, and finally we describe and analyse an algorithm for digraphs of average outdegree at most d.

3.1 Preliminaries

Interfaces and switching walks. Let G be a directed graph (digraph). For a vertex v a set I_v^{in} of all incoming edges to v or a set I_v^{out} of all outgoing edges from v will be called an *interface* of v. We define the type of an interface of v so that $\mathsf{type}(I_v^{\mathsf{in}}) = \mathsf{in}$ and $\mathsf{type}(I_v^{\mathsf{out}}) = \mathsf{out}$.

Consider a sequence of distinct edges $\pi = e_1, \ldots, e_k$ in G such that if we forget about the orientation of edges, then we get a walk $v_1, \ldots, v_{k+1}$ in the underlying undirected graph, where for $i = 1, \ldots, k$ edge e_i is an orientation of $v_i v_{i+1}$. Assume additionally that for every $i = 2, \ldots, k$ either both edges e_{i-1} and e_i enter v_i or both leave v_i, in other words, the orientation of edges on the walk alternates. Now, let $I_1, \ldots, I_{k+1}$ be the consecutive interfaces visited by π, i.e., for every $j = 1, \ldots, k+1$ we have that I_j is an interface of v_j and for every $j = 1, \ldots, k$, we have $e_j \in I_j \cap I_{j+1}$. If $|I_1|, |I_k| > 2$ and $|I_j| = 2$, for $j = 2, \ldots, k-1$, the sequence π will be called a *switching walk*. Similarly, if $|I_j| = 2$ for $j = 1, \ldots, k$, and $v_1 = v_{k+1}$, i.e., the walk $v_1, \ldots, v_{k+1}$ is closed, then π will be called a *switching circuit*. In both cases, *length* of π is defined as k. The sequence $v_1, \ldots, v_{k+1}$ is called the *vertex sequence* of π. Abusing the notation slightly, we will refer to π as a set, when it is convenient. The motivation for introducing the notions of switching walks and circuits is given by the following lemma.

▶ **Lemma 3.1.** *Let $\pi = \{e_1, \ldots, e_k\}$ be a switching walk or a switching circuit in a digraph G. Let $H \subseteq E(G)$ be a Hamiltonian cycle in G. Then, $H \cap \pi = \{e_{2i-1} \mid i = 1, \ldots, \lfloor \frac{k+1}{2} \rfloor\}$, or $H \cap \pi = \{e_{2i} \mid i = 1, \ldots, \lfloor \frac{k}{2} \rfloor\}$.*

Proof. Let us assume that π is a switching walk. (For a switching circuit the proof is analogous.) Consider two consecutive edges $e_i, e_{i+1} \in \pi$. By the definition of a switching walk, there is a vertex v with an interface I of size 2 such that $I = \{e_i, e_{i+1}\}$. Since the cycle H passes through v, we obtain that H must contain exactly one of the edges e_i and e_{i+1}, and the lemma easily follows. ◀

In some cases it is convenient to study switching walks and circuits in the language of an auxiliary bipartite graph. Let $V^{\mathsf{out}} = \{v^{\mathsf{out}} \mid v \in V(G)\}$ and $V^{\mathsf{in}} = \{v^{\mathsf{in}} \mid v \in V(G)\}$. The *interface graph of* G is the bipartite graph I_G such that $V(I_G) = V^{\mathsf{out}} \cup V^{\mathsf{in}}$ and $E(I_G) = \{u^{\mathsf{out}}v^{\mathsf{in}} \mid (u,v) \in E(G)\}$. Clearly, there is a one-to-one correspondence between interfaces in G and vertices of I_G, and the degree of a vertex in I_G is the size of the corresponding interface. Moreover, if $\pi = e_1, \ldots, e_k$ is a switching walk in G with a vertex sequence $v_1, \ldots, v_{k+1}$ and interface sequence $I_1, \ldots, I_{k+1}$, then π corresponds to a simple path $I(\pi) = v_1^{\mathsf{type}(I_1)}, \ldots, v_{k+1}^{\mathsf{type}(I_{k+1})}$ in G with endpoints of degree larger than 2, and all inner vertices of degree 2. Similarly, a switching circuit π corresponds to a simple cycle $I(\pi)$ in I_G with all vertices of degree 2 in I_G, i.e., $I(\pi)$ forms a connected component in I_G. Observe that both in the case of path and cycle above, the edges $I(\pi)$ are exactly the edges of I_G corresponding to the edges of π. Using the equivalence described in this paragraph, the following lemma is immediate.

▶ **Lemma 3.2.** *Edges of every digraph can be uniquely partitioned into switching walks and circuits. Moreover, the partition can be computed in linear time.*

Proof. Let G be a digraph. Recall that by the definition of I_G, there is a one-to-one correspondence between edges of G and edges of I_G. It is clear that edges of I_G can be uniquely partitioned into (1) cycles with all vertices of degree 2 and (2) paths with both endpoints of degree at least 3 and all inner vertices of degree 2. The corresponding switching circuits and switching walks form the desired partition of $E(G)$. An algorithm which constructs the partition is straightforward. ◀

3.2 Branching subroutine

Let us consider a digraph G. By $t_i(G)$ we will denote the number of vertices of G with outdegree equal to i. Let $k = n - t_1(G)$ be the number of vertices of G with outdegree at least 2, and let $s_1, \ldots, s_k$ be the sequence of these outdegrees. Then, let us denote the sum $\sum_{i=1}^{k}(s_i - 2)$ by $S(G)$. An analogous sum for indegrees will be denoted by $S^-(G)$. Note that if G has no vertex of out- or indegree 1, then by the handshaking lemma $S(G) = S^-(G)$.

▶ **Theorem 3.3.** ATSP *can be solved in time* $\mathcal{O}^*(2^{(n-t_1(G))/3 \,+\, \beta S(G)})$ *and polynomial space, where* $\beta = \log_2 3 - 1 < 0.585$*.*

Proof. The idea behind this algorithm is to branch on interfaces of size greater than 2, reducing the initial problem to the case of $(2,2)$-graphs, and then to apply Corollary 2.6. A detailed description is presented in Pseudocode 1. Our algorithm consists of two functions: $\mathsf{AtspBranching}(G, \mathsf{weight})$ – the main one, which solves ATSP in G, and an auxiliary function $\mathsf{AtspForcedEdge}(G, \mathsf{weight}, e)$ that returns the minimum weight of a Hamiltonian cycle H in G such that $e \in H$ (or ∞ if there is no such cycle). Note that $\mathsf{AtspForcedEdge}$ modifies the input digraph G, and calls $\mathsf{AtspBranching}$ on the new digraph G'. We observe that every Hamiltonian cycle in G' of weight w corresponds to a Hamiltonian cycle in G of weight $w + \mathsf{weight}(e)$ and containing edge e, and vice versa.

Given a digraph G with a function $\mathsf{weight} : E(G) \to \mathbb{Z}$, $\mathsf{AtspBranching}$ starts by considering a number of trivial cases $(a)-(c)$, where either G has only 2 vertices, or there is a vertex with out- or indegree at most 1. Next, we apply Lemma 3.2 to decompose $E(G)$ into switching walks and circuits, and we deal with a situation when there is a switching walk $\pi = (e_1, \ldots, e_{2k})$ of even length in G (cases $(d)-(e)$ in Pseudocode 1). Denote by I, respectively I', the interface which π starts, respectively ends, at. Consider a Hamiltonian cycle H in G. By Lemma 3.1 we obtain that either $e_1 \in H \cap \pi$, or $e_{2k} \in H \cap \pi$. We consider the following two cases.

Algorithm 1 AtspBranching(G, weight).

```
Input: G – a digraph on n ≥ 2 vertices,
       weight – a function E(G) → ℤ
Output: the minimum weight of a Hamiltonian cycle in G,
        or ∞ if there is no such cycle

Function AtspForcedEdge(G, weight, e):
    Let e = (u, v)
    G_1 ← G with removed edges of the form (v, u), (u, x) and (x, v) for x ∈ V(G)
    G' ← G_1 with contracted vertices u and v
    weight' ← weights of E(G') inherited from G appropriately
    return weight(e) + AtspBranching(G', weight')

Function AtspBranching(G, weight):
    if G has exactly two vertices u and v then                                   (a)
        return weight((u, v)) + weight((v, u)) if (u, v), (v, u) ∈ E(G), or ∞ otherwise
    if there is an empty interface in G i.e. a vertex of out- or indegree 0 then (b)
        return ∞
    if there is an interface I = {e} of size 1 then                             (c)
        return AtspForcedEdge(G, weight, e)
    Use Lemma 3.2 to partition E(G) into switching walks and circuits
    if there is a switching walk π which begins and ends at the same interface I then (d)
        G' ← G with removed edges of I \ π
        return AtspBranching(G', weight)
    if there is a switching walk π of even length then                          (e)
        Let π = (e_1, ..., e_2k)
        return min(AtspForcedEdge(G, weight, e_1), AtspForcedEdge(G, weight, e_2k))
    if there is no interface of size at least 3 then                            (f)
        Apply Corollary 2.6 to G and return the weight of the solution, or ∞
    else                                                                        (g)
        Let I = {e_1, ..., e_s} be an out-interface of size s ≥ 3
        result ← ∞
        for i = 1, ..., s do
            result ← min(result, AtspForcedEdge(G, weight, e_i))
        return result
```

- If $I = I'$, then we have $H \cap I \in \{e_1, e_{2k}\}$, and thus all edges of $I \setminus \pi$ can be safely removed as they cannot be extended to a Hamiltonian cycle in G. This is realized in step (d) of the pseudocode. Note that if a switching walk π starts and ends at the same interface, then it must be of even length, since orientation of edges on π alternates.
- If $I \neq I'$, we branch by guessing if $e_1 \in H \cap \pi$, or $e_{2k} \in H \cap \pi$ (step (e) of the pseudocode).

If none of the above cases holds, we check whether all interfaces consist of at most 2 edges (cases $(f) - (g)$ in Pseudocode 1). If so, then G is a $(2, 2)$-graph, and we can solve ATSP for G by applying Corollary 2.6. If not, we choose an out-interface I of size at least 3, and we branch on it, by guessing which of the edges of I to pick as a part of a Hamiltonian cycle. Note that since G has no interface of size 1, then it has an interface of size at least 3 if and only if it has an out-interface of size at least 3.

Time complexity analysis. We begin with providing a few simple facts concerning the properties of our algorithm.

▷ Claim 3.4 (♠). During execution of algorithm AtspBranching, the value of $S(G)$ cannot increase.

▷ Claim 3.5 (♠). During execution of algorithm AtspBranching, graph G is simple, i.e. does not contain two edges of the same head and tail.

▷ Claim 3.6. Let $\pi = (e_1, \ldots, e_k)$ be a switching walk in G. Assume that during the run of our algorithm we decided to take an edge e_1 by calling $\mathsf{AtspForcedEdge}(G, \mathsf{weight}, e_1)$. Then, by exhaustively applying rule (c) of AtspBranching to the resulting digraph, we will remove from G all edges of the form e_{2i}, and contract all edges of the form e_{2i+1}. An analogous statement can be made if we start with discarding edge e_1 instead of contracting it.

Denote $f(n, S) = 2^{n/3+\beta S}$, where β is the constant from Theorem 3.3. We need to prove that the running time of our algorithm is bounded by $f(n - t_1(G), S(G))n^{\mathcal{O}(1)}$. We proceed by induction on $t_1(G) + S(G)$.

If $t_1(G) > 0$, then our algorithm starts by choosing edges which form interfaces of size 1, what leads to a digraph with at most $\max(2, n - t_1(G))$ vertices. Hence, by the induction hypothesis the running time is bounded by $f(n - t_1(G), S(G))n^{\mathcal{O}(1)}$.

In what follows we assume $t_1(G) = 0$. If G satisfies condition (a) or (b), then our algorithm runs in polynomial time. Similarly, we can assume that G does not satisfy conditions (c) and (d), as applying the corresponding reductions exhaustively takes only polynomial time and does not increase the value of $S(G)$, according to Claim 3.4.

From now on, we assume that conditions $(a) - (d)$ do not hold for G. If $S(G) = 0$, then our algorithm executes the algorithm from Corollary 2.6 and therefore its running time is bounded by $\mathcal{O}^*(2^{n/3})$, as desired. Now, assume $S(G) > 0$. It remains to analyse cases (e) and (g) of AtspBranching.

Case (e). Let us assume that there is a switching walk $\pi = (e_1, \ldots, e_{2k})$ of even length in G which starts at interface I of size $s \geq 3$, and ends at interface $I' \neq I$ of size $s' \geq 3$. Let G' be a digraph obtained from G by running $\mathsf{AtspForcedEdge}(G, \mathsf{weight}, e_1)$ and exhaustively applying rules $(a) - (d)$ to the resulting digraph.

Since edge e_1 is contracted in AtspForcedEdge, we have $|V(G')| \leq |V(G)| - 1$. We claim that $S(G') \leq S(G) - 2$. Assume $\mathsf{type}(I) = \mathsf{type}(I') = \mathsf{out}$. By Claim 3.6, for all $i = 1, \ldots, k$, edge e_{2i-1} was contracted, and edge e_{2i} was removed. We observe that contracting edge e_1 results in removing interface I from the graph, and discarding edge e_{2k} decreases the size of I' by 1. By Claim 3.4 operations performed on edges $e_2, \ldots, e_{2k-1}$ do not increase the value of $S(G)$. Hence, $S(G) - S(G') \geq (s - 2) + 1 \geq 2$, as desired. If $\mathsf{type}(I) = \mathsf{type}(I') = \mathsf{in}$, then by the same reasoning, we obtain $S^-(G') \leq S(G) - 2$ but since there are no interfaces of size 1 in G', we have $S(G') = S^-(G')$, and the claim follows.

Hence, by the induction hypothesis, the running time of our algorithm applied to G' is bounded by $f(n-1, S(G)-2)$. To obtain the desired bound for digraph G we need to show that $2f(n-1, S(G)-2) \leq f(n, S(G))$, or, equivalently $\log_2(2f(n-1, S(G)-2)) \leq \log_2 f(n, S(G))$. We obtain

$$\begin{aligned}\log_2(2f(n-1, S(G)-2)) &= 1 + \tfrac{n-1}{3} + \beta(S(G) - 2) = \tfrac{n}{3} + \beta S(G) + \tfrac{2}{3} - 2\beta \\ &\leq \tfrac{n}{3} + \beta S(G) = \log_2 f(n, S(G)).\end{aligned}$$

Case (g). Now, we assume that G does not satisfy conditions $(a) - (f)$. Let I be an out-interface of size $s \geq 3$, and consider an edge $e \in I$. Let G' be a digraph obtained from G after choosing edge e by running $\mathsf{AtspForcedEdge}(G, \mathsf{weight}, e)$, and let G'' be a digraph obtained from G' by the subsequent exhaustive application of rules $(a) - (d)$ by $\mathsf{AtspBranching}$. Define $\Delta n = |V(G)| - |V(G'')|$, and $\Delta S = S(G) - S(G'')$.

▷ Claim 3.7. It holds that $\Delta n \geq 1$, $\Delta S \geq s - 2 \geq 1$, and $\Delta n + \Delta S \geq s + 1$.

Proof. For a digraph G we denote $n(G) = |V(G)|$. First, we analyse a direct impact of calling $\mathsf{AtspForcedEdge}(G, \mathsf{weight}, e)$. All edges of I are removed from G, hence by Claim 3.4 we have $\Delta S \geq S(G) - S(G') \geq s - 2 \geq 1$. Moreover, edge e gets contracted, and thus $\Delta n \geq n(G) - n(G') = 1$. We are left with proving that $(n(G') - n(G'')) + (S(G') - S(G'')) \geq 2$, since then we will have $\Delta n + \Delta S \geq s + 1$.

Let π be the switching walk which starts with edge e. Let e' be the last edge of π (it is possible that π has length 1 and $e' = e$). We recall that at step (g) every switching walk in G is of odd length. Take an in-interface I' such that $e' \in I'$. By the definition of switching walk, $|I'| \geq 3$, so let e', e'_1, e'_2 be three different edges of I'. For $j = 1, 2$ denote by π_j the switching walk which ends with edge e'_j. Let e_j be the first edge of π_j, and let I_j be an out-interface such that $e_j \in I_j$.

Let $F, R \subseteq E(G)$ be edges of G which correspond to the edges that were taken (and hence, contracted) and removed, respectively, during the run of our algorithm which leads from digraph G to digraph G''. We have $e \in F$. By Claim 3.6 applied to π, we obtain $e' \in F$. Therefore, $e'_1, e'_2 \in R$, and again by Claim 3.6 applied to π_1 and π_2, we obtain $e_1, e_2 \in R$. Now, we consider a few cases.

- If I, I_1, I_2 are pairwise different out-interfaces, then during processing of digraph G' we removed edges e_1, e_2 from different out-interfaces of size at least 3. Therefore, $S(G') - S(G'') \geq 2$.
- If $I = I_1 = I_2$, then among switching walks π, π_1, π_2 there are least two of length greater than 1 (hence, of length at least 3), because otherwise the graph is not simple, contradicting Claim 3.5. Let us assume that these are walks π and π_1 (the other cases are analogous). Then, by Claim 3.6, during processing of digraph G' we contracted edge e' and the second edge of π_1. Therefore, $n(G') - n(G'') \geq 2$.
- If $I_1 = I_2 \neq I$, or $I = I_1 \neq I_2$, or $I = I_2 \neq I_1$, then at least one switching walk among π, π_1, π_2 is of length at least 3, and there is another interface apart from I that gets smaller. Hence, we obtain in a similar way as before that $n(G'') - n(G') \geq 1$, and $S(G'') - S(G') \geq 1$. ◁

Since $\Delta S \geq 1$, we have $S(G'') < S(G)$, and thus by the induction hypothesis the running time of our algorithm applied to G'' is bounded by $f(n(G''), S(G'')) = f(n - \Delta n, S(G) - \Delta S)$. In step (g) of $\mathsf{AtspBranching}$ we branch into s such subcases, hence we need to prove that $s \cdot f(n - \Delta n, S(G) - \Delta S)) \leq f(n, S(G))$. We will show the equivalent $\log_2(s \cdot f(n - \Delta n, S(G) - \Delta S)) \leq \log_2 f(n, S(G))$. Indeed,

$$
\begin{aligned}
&\log_2(s \cdot f(n - \Delta n, S(G) - \Delta S)) \\
&\qquad = \log_2 s + \tfrac{n - \Delta n}{3} + \beta(S(G) - \Delta S) \\
&\qquad = \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{\Delta n}{3} - \beta \Delta S \\
&\qquad \leq \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{s + 1 - \Delta S}{3} - \beta \Delta S && \text{(Claim 3.7)} \\
&\qquad = \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{s+1}{3} - (\beta - \tfrac{1}{3})\Delta S \\
&\qquad \leq \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{s+1}{3} - (\beta - \tfrac{1}{3})(s - 2) && \text{(Claim 3.7)}
\end{aligned}
$$

$$
\begin{aligned}
&= \tfrac{n}{3} + \beta S(G) + \log_2 s - 1 - \beta(s-2) \\
&\le \tfrac{n}{3} + \beta S(G) && (\triangle) \\
&= \log_2 f(n, S(G)).
\end{aligned}
$$

where inequality $(\triangle)$ follows from the fact that the function $x \mapsto \frac{\log_2 x - 1}{x-2}$ is decreasing on $[3, \infty)$, and thus it can be bounded by the value at $x = 3$ which is equal to β. Consequently, the inequality $\log_2 s \le 1 + \beta(s-2)$ holds for $s \ge 3$. ◀

3.3 General algorithm

The idea behind our general algorithm is to run in parallel two algorithms: our branching algorithm from Theorem 3.3 (which we will refer to as Algorithm Ⓐ), and enumerating cycle covers from Subsection 2.2 (Algorithm Ⓑ) . We finish when one of these algorithms terminates. Our goal is to prove that the time complexity of such an approach is bounded by $\mathcal{O}^*(2^{\alpha(d-1)n})$ if we apply it to digraphs of average outdegree at most d, where α is the constant from Theorem 1.1. (Note that when implementing this algorithm, one may also compare the values of $\frac{n}{3} + \beta S(G)$ and $\alpha(d-1)n$, and, depending on the result, run either Algorithm Ⓐ, or Algorithm Ⓑ.) For the time complexity analysis, see the full version.

4 Exponential space algorithm

In this section we establish Theorem 1.2.

Let G be a digraph with n vertices and $m = dn$ edges. For simplicity, we assume in this section that n is even, for otherwise we can pick an arbitrary vertex v, and split it into two vertices v^{in} and v^{out} with edges inherited from v appropriately and with one additional edge $(v^{\mathsf{in}}, v^{\mathsf{out}})$ – this operation adds one vertex to the graph but does not increase the average outdegree. We will say that a simple path P in G is (l, D)-*light* if the length of P is l, and the sum of outdegrees of inner vertices of P is bounded by D. For a vertex $v \in V(G)$, and positive integers l, D, by $\mathcal{P}_{v,l,D}$ we will denote the family of all (l, D)-light paths in G which start at vertex v.

Our algorithm relies on the following two lemmas.

▶ **Lemma 4.1.** *Let H be a Hamiltonian cycle in G. Then, the edges of H can be partitioned into two $(n/2,\ m/2)$-light paths.*

▶ **Lemma 4.2.** *For a digraph G, a vertex v, and integers l, D, the family $\mathcal{P}_{v,l,D}$ can be computed in time $\tau(D/(l-1))^{l-1} n^{\mathcal{O}(1)}$ where the function τ is defined as in Theorem 1.2.*

Before we proceed to the proofs of above lemmas, let us see how to derive Theorem 1.2 from them. Given a digraph G, the algorithm starts by iterating over all pairs of distinct vertices u_1 and u_2. For each such a pair we use Lemma 4.2 to obtain the families $\mathcal{P}_1 = \mathcal{P}_{u_1,n/2,m/2}$ and $\mathcal{P}_2 = \mathcal{P}_{u_2,n/2,m/2}$. By filtering them, we may assume that all paths from $\mathcal{P}_1$ end at u_2, and all paths from $\mathcal{P}_2$ end at u_1. Next, we create a dictionary $\mathcal{D}$ with an entry $\{\text{key} : V(P_1), \text{value} : \text{weight}(P_1)\}$ for every path $P_1 \in \mathcal{P}_1$. (In case there is more than one path on the same set of vertices we keep only one entry with the minimum weight.) Then, we iterate over all paths $P_2 \in \mathcal{P}_2$, and we look up in $\mathcal{D}$ a subset $V'(P_2) := (V(G) \setminus V(P_2)) \cup \{u_1, u_2\}$. For every hit we calculate the sum: $\text{weight}(P_2) + \mathcal{D}[V'(P_2)]$, and we return the minimum of these values.

Algorithm 2 GeneratePaths(G, path, l, D).

Input: G – a digraph,
path – a sequence of vertices forming a path in G,
l, D – positive integers
Output: A collection of all simple paths of the form: path#$(v_1, \dots, v_l)$ such that $\sum_{i=1}^{l-1} \text{outdeg}(v_i) \leq D$

```
u ← the last vertex on path;
for vertex v_1 such that (u, v_1) ∈ E(G) and v_1 ∉ path do
    if l = 1 then
        print path#v_1;
    else if outdeg(v_1) + (l − 2) ≤ D then                        (✓)
        GeneratePaths(G, path#v_1, l − 1, D − outdeg(v_1));
```

The correctness of this procedure is a direct corollary from Lemma 4.1. Moreover, the running time of the algorithm is dominated, up to a polynomial factor, by the running time of the algorithm from Lemma 4.2, which in our case is bounded by

$$\tau\left(\frac{\frac{m}{2}}{\frac{n}{2}-1}\right)^{n/2-1} n^{\mathcal{O}(1)} = \tau\left(\frac{d}{1-\frac{2}{n}}\right)^{n/2-1} n^{\mathcal{O}(1)} = \tau(d)^{n/2} n^{\mathcal{O}(1)}$$

where the last equality follows from the fact that when d is fixed, then for sufficiently large n we have $\lfloor d/(1-\frac{2}{n})\rfloor = \lfloor d \rfloor$. Note that we implement the dictionary $\mathcal{D}$ as a balanced tree, so each lookup takes time $\mathcal{O}(\log|\mathcal{D}|) = \mathcal{O}(n)$.

Proof of Lemma 4.1. Let $k = n/2$, and let $d_0, d_1, \dots, d_{2k-1}$ be the outdegrees of consecutive vertices on H. Denote $S_i = d_i + d_{i+1} + \ldots + d_{i+k-1}$. (In this proof indices are understood modulo $2k$.) We need to prove that for some index j both expressions $S_j - d_j$ and $S_{j+k} - d_{j+k}$ do not exceed $m/2$.

Let $R_i := S_i - S_{i+k}$. We observe that $R_k = S_k - S_0 = -R_0$. Hence, there exists an index $j \in \{0, \dots, k-1\}$ such that $R_j \cdot R_{j+1} \leq 0$. Without loss of generality, we may assume that $R_j \leq 0$ (equivalently, $S_j \leq S_{j+k}$), for otherwise we can just shift all indices by k. Then, $R_{j+1} \geq 0$ (equivalently, $S_{j+1+k} \leq S_{j+1}$). Thus we obtain

$$\begin{aligned} S_j - d_j &\leq S_j \leq \tfrac{1}{2}(S_j + S_{j+k}) = \tfrac{m}{2} \\ S_{j+k} - d_{j+k} &\leq S_{j+k+1} \leq \tfrac{1}{2}(S_{j+k+1} + S_{j+1}) = \tfrac{m}{2}. \end{aligned}$$

This ends the proof. ◀

Before we proceed to the proof of Lemma 4.2, we state a technical lemma.

▶ **Lemma 4.3** (♠). *Let $a_1, \dots, a_k$ be integers with an average bounded by $\bar{a}$. Then, $a_1 \cdot \ldots \cdot a_k \leq \tau(\bar{a})^k$.*

Proof of Lemma 4.2. We apply a simple branching procedure which starts at vertex v, and at each step guesses the next vertex on a path by considering all *reasonable* possibilities. A detailed description of the algorithm can be found in Pseudocode 2. (To compute the family $\mathcal{P}_{v,l,D}$ we call the function GeneratePaths with the arguments $(G, \{v\}, l, D)$.) Note that before appending a vertex to the current path we check whether the sum of outdegrees on the new path is not too large (line marked with (✓) in the Pseudocode). More precisely, we check whether appending a sequence of vertices of outdegree 1 to the new path would give us a correct (l, D)-path.

The correctness of such a procedure is straightforward. It remains to estimate its time complexity. Let $\mathcal{T}(G)$ be a search tree representing execution of this algorithm. We claim that $\mathcal{T}(G)$ contains at most $\tau(D/(l-1))^{l-1}n$ leaves, where $\tau(d) = \lfloor d\rfloor^{\lfloor d\rfloor+1-d}(\lfloor d\rfloor+1)^{d-\lfloor d\rfloor} \leq d$

We will say that a directed rooted tree $\mathcal{T}$ has the property ($\star$) if for any path $u_0, \dots, u_h$ from the root of $\mathcal{T}$ to some leaf u_h we have $h \leq l$, and the sum $\sum_{i=1}^{h-1} \operatorname{outdeg}(u_i)$ is bounded by $D-(l-h)$. From the description of the algorithm we see that $\mathcal{T}(G)$ has the property ($\star$). Indeed, let $u_0, \dots, u_h$ be a path from the root to a leaf in $\mathcal{T}(G)$. Before the algorithm entered the vertex u_{h-1} the following condition was checked:

$$\operatorname{outdeg}(u_{h-1}) + (l-(h-2)-2) \leq D - \sum_{i=1}^{h-2} \operatorname{outdeg}(u_i)$$

which is equivalent to $\sum_{i=1}^{h-1} \operatorname{outdeg}(u_i) \leq D-(l-h)$. Hence it is enough to prove the following claim. (A similar statement appears in the work of Gebauer [19].)

▷ Claim 4.4. Any tree $\mathcal{T}$ with the property ($\star$) has at most $\tau(D/(l-1))^{l-1}n$ leaves.

Given a tree $\mathcal{T}$ with the property ($\star$) we modify it so that the property ($\star$) is preserved and the number of leaves in it does not increase. First, we may assume that all leaves in $\mathcal{T}$ are at depth exactly l. Indeed, let $u_0, \dots, u_h$ be a path from the root of $\mathcal{T}$ to some leaf u_h at depth $h < l$. Then, we may append to it a path $u_h, u_{h+1}, \dots, u_l$ – this operation does not change the number of leaves, and the property ($\star$) is preserved because

$$\sum_{i=1}^{l-1} \operatorname{outdeg}(u_i) = \sum_{i=1}^{h-1} \operatorname{outdeg}(u_i) + (l-h) \leq D$$

Next, we modify $\mathcal{T}$ iteratively. Let $\mathcal{T}_1 := \mathcal{T}$. At i-th step, for $i = 1, \dots, l-1$, we consider the family $\mathcal{S}_i$ of all subtrees in $\mathcal{T}_i$ with a root at depth i. Let $S_i \in \mathcal{S}_i$ be a subtree with the maximum number of leaves. We create a tree $\mathcal{T}_{i+1}$ by substituting in $\mathcal{T}_i$ all subtrees from $\mathcal{S}_i$ with S_i. We observe that for every $i = 1, \dots, l-1$ tree $\mathcal{T}_i$ has depth l, the number of leaves in $\mathcal{T}_i$ is bounded by the number of leaves in $\mathcal{T}_{i+1}$, and all vertices in $\mathcal{T}_i$ at the same depth $j \leq i-1$ have the same outdegree. Combining the latter property with the fact that the condition ($\star$) holds for leaves in the subtree S_i, we obtain inductively that every tree $\mathcal{T}_i$ still has the property ($\star$).

Now, we consider the tree $\mathcal{T}_l$. For $i = 0, \dots, l-1$ let d_i be the outdegree of any vertex at depth i in $\mathcal{T}_l$. Then we may bound the number of leaves in $\mathcal{T}_l$ by

$$d_0 \cdot \prod_{i=1}^{l-1} d_i \leq n \left(\frac{\sum_{i=1}^{l-1} d_i}{l-1}\right)^{l-1} \leq n \left(\frac{D}{l-1}\right)^{l-1}$$

To obtain a tighter bound on the size of $\mathcal{T}_l$ we observe that in the above estimation we obtain an equality only if $d_i = D/(l-1)$ for $i = 1, \dots, l-1$. However, this is impossible unless expression $D/(l-1)$ is integral. After applying Lemma 4.3 we get the tighter bound which proves the claim of Lemma 4.2. ◀

References

1 Eric T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Inf. Process. Lett.*, 47(4):203–207, September 1993. doi:10.1016/0020-0190(93)90033-6.

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

3 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.

4 Andreas Björklund. Exploiting sparsity for bipartite hamiltonicity. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:11, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ISAAC.2018.3.

5 Andreas Björklund. An asymptotically fast polynomial space algorithm for hamiltonicity detection in sparse directed graphs, 2020. arXiv:2009.11780.

6 Andreas Björklund and Thore Husfeldt. The parity of directed Hamiltonian cycles. In *FOCS*, pages 727–735. IEEE Computer Society, 2013.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The traveling salesman problem in bounded degree graphs. *ACM Trans. Algorithms*, 8(2):18:1–18:13, 2012. doi:10.1145/2151171.2151181.

8 Andreas Björklund, Vikram Kamat, Lukasz Kowalik, and Meirav Zehavi. Spotting trees with few leaves. *SIAM J. Discret. Math.*, 31(2):687–713, 2017. doi:10.1137/15M1048975.

9 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, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 91:1–91:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.91.

10 Andreas Björklund and Ryan Williams. Computing permanents and counting Hamiltonian cycles by listing dissimilar vectors. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 25: 1–25: 14. Schloss Dagstuhl - Leibniz Center for Computer Science, 2019. doi:10.4230/LIPIcs.ICALP.2019.25.

11 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

12 Lev Meerovich Brègman. Some properties of nonnegative matrices and their permanents. *Doklady Akademii Nauk*, 211(1):27–30, 1973.

13 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.

14 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 Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

15 Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. *Inf. Comput.*, 243:75–85, 2015. doi:10.1016/j.ic.2014.12.007.

16 David Eppstein. The traveling salesman problem for cubic graphs. *J. Graph Algorithms Appl.*, 11(1):61–81, 2007. doi:10.7155/jgaa.00137.

17 Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009. doi:10.1007/s00453-007-9133-3.

18 Komei Fukuda and Tomomi Matsui. Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters*, 7(1):15–18, 1994.

19 Heidi Gebauer. How many Hamilton cycles and perfect matchings are there? Master's thesis, ETH Zürich, March 2007.

20 Heidi Gebauer. On the number of Hamilton cycles in bounded degree graphs. In Robert Sedgewick and Wojciech Szpankowski, editors, *Proceedings of the Fifth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2008, San Francisco, California, USA, January 19, 2008*, pages 241–248. SIAM, 2008. doi:10.1137/1.9781611972986.8.

21 Heidi Gebauer. Finding and enumerating Hamilton cycles in 4-regular graphs. *Theor. Comput. Sci.*, 412(35):4579–4591, 2011. doi:10.1016/j.tcs.2011.04.038.

22 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.

23 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.

24 Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1(2):49–51, April 1982. doi:10.1016/0167-6377(82)90044-X.

25 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. A bound on the pathwidth of sparse graphs with applications to exact algorithms. *SIAM J. Discret. Math.*, 23(1):407–427, 2009. doi:10.1137/080715482.

26 Samuel Kohn, Allan Gottlieb, and Meryle Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the 1977 Annual Conference*, ACM '77, page 294–300, New York, NY, USA, 1977. Association for Computing Machinery. doi:10.1145/800179.810218.

27 Łukasz Kowalik and Konrad Majewski. The asymmetric travelling salesman problem in sparse digraphs, 2020. arXiv:2007.12120.

28 Jesper Nederlof. Bipartite TSP in $o(1.9999^n)$ time, assuming quadratic time matrix multiplication. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 40–53. ACM, 2020. doi:10.1145/3357713.3384264.

29 Mohd Shahrizan Othman, Aleksandar Shurbevski, and Hiroshi Nagamochi. Polynomial-space exact algorithms for the bipartite traveling salesman problem. *IEICE Trans. Inf. Syst.*, 101-D(3):611–612, 2018. doi:10.1587/transinf.2017FCL0003.

30 Ján Plesník. The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two. *Inf. Process. Lett.*, 8(4):199–201, 1979. doi:10.1016/0020-0190(79)90023-1.

31 Frank Rubin. A search procedure for Hamilton paths and circuits. *J. ACM*, 21(4):576–580, October 1974. doi:10.1145/321850.321854.

32 Alexander Schrijver. A short proof of Minc's conjecture. *Journal of combinatorial theory, Series A*, 25(1):80–83, 1978.

33 Mingyu Xiao and Hiroshi Nagamochi. An exact algorithm for TSP in degree-3 graphs via circuit procedure and amortization on connectivity structure. *Algorithmica*, 74(2):713–741, 2016. doi:10.1007/s00453-015-9970-4.

34 Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for TSP in graphs of maximum degree 4. *Theory Comput. Syst.*, 58(2):241–272, 2016. doi:10.1007/s00224-015-9612-x.

35 Norhazwani Md Yunos, Aleksandar Shurbevski, and Hiroshi Nagamochi. An improved-time polynomial-space exact algorithm for TSP in degree-5 graphs. *J. Inf. Process.*, 25:639–654, 2017. doi:10.2197/ipsjjip.25.639.