Bipartite TSP in $O(1.9999^n)$ Time, Assuming Quadratic Time Matrix Multiplication

Jesper Nederlof*
j.nederlof@uu.nl
Utrecht University
Utrecht, The Netherlands

ABSTRACT

The symmetric traveling salesman problem (TSP) is the problem of finding the shortest Hamiltonian cycle in an edge-weighted undirected graph. In 1962 Bellman, and independently Held and Karp, showed that TSP instances with n cities can be solved in $O(n^2 2^n)$ time. Since then it has been a notorious problem to improve the runtime to $O((2-\varepsilon)^n)$ for some constant $\varepsilon > 0$. In this work we establish the following progress: If $(s \times s)$-matrices can be multiplied in $s^{2+o(1)}$ time, than all instances of TSP in *bipartite* graphs can be solved in $O(1.9999^n)$ time by a randomized algorithm with constant error probability. We also indicate how our methods may be useful to solve TSP in non-bipartite graphs.

On a high level, our approach is via a new problem called MinHamPair: Given two families of weighted perfect matchings, find a combination of minimum weight that forms a Hamiltonian cycle. As our main technical contribution, we give a fast algorithm for MinHamPair based on a new sparse cut-based factorization of the 'matchings connectivity matrix', introduced by Cygan et al. [JACM'18].

CCS CONCEPTS

• **Theory of computation → Parameterized complexity and exact algorithms**.

KEYWORDS

Traveling Salesman Problem, Exponential Time algorithms

ACM Reference Format:
Jesper Nederlof. 2020. Bipartite TSP in $O(1.9999^n)$ Time, Assuming Quadratic Time Matrix Multiplication. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20), June 22–26, 2020, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3357713.3384264

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is one of the favorite and most well-studied computational problems in theoretical computer science. Its simple statement and clear applicability – a salesman needs to find the shortest tour through n different cities – make the problem extremely attractive. From a theoretical perspective, the problem poses crisp yet very challenging research challenges that make it representative for many NP-complete problems.

As an example, improving the famous polynomial time 1.5-approximation algorithm by Christofides [Chr76] (and independently, by Serdyukov [Ser78]) is one of the favorite open questions in the theory of approximation algorithms. While this question remains open, recent years witnessed a plethora of breakthroughs related to approximating TSP solutions such as the first constant factor approximation for *asymmetric*[1] TSP (ATSP) with triangle inequality by Svensson et al. [STV18] and an algorithm generalizing the 1.5-approximation to the more general path-TSP by Traub and Vygen [TV19] and Zenklusen [Zen19].

Another, perhaps even more classic, line of research is to solve TSP *exactly* as fast as possible in the worst case. Of course we do not expect to solve the problem in polynomial time since its decision variant is NP-complete, but nevertheless it is of interest to determine how fast it can be solved. This study was initiated already in the 1920's by Menger (see [Sch05] for an historial account). In 1962, Bellman [Bel62] and, independently, Held and Karp [HK62] proposed a Dynamic Programming (DP) algorithm with table entries defined for each subset of the vertices to build partial tours. We quote the following excerpt from the popular text book *'In Pursuit of the Traveling Salesman'* about this result:

> *This team's dynamic programming algorithm solves any instance in time proportial to $n^2 2^n$, and this is where we still stand, after nearly fifty years. A revolution may be overstating what is needed to push beyond Held-Karp, but it clearly is going to take an exciting new idea.*
> — *William Cook* [Coo11]

As also formalized in [Coo11], the research challenge can be more precisely stated as follows:

> **Open Question 1:** Can (A)TSP on n cities be solved in $O((2-\varepsilon)^n)$ time, for some constant $\varepsilon > 0$?

A positive answer to Open Question 1 would be valuable because of several reasons: First, generally speaking, 'improved algorithms' such as the one asked here would indicate that relatively simple approaches can be improved and thus that the computational landscape of worst-case complexity of NP-complete problems is even more complex than we may think. Second, such improved algorithms are highly relevant for the blooming area of *Fine-Grained*

*Part of this work was conducted while the author worked at Eindhoven University of Technology. Supported the European Research Council under project no. 853234.

STOC '20, June 22–26, 2020, Chicago, IL, USA

ACM ISBN 978-1-4503-6979-4/20/06…$15.00
https://doi.org/10.1145/3357713.3384264

[1] Meaning the distance from i to j can be different from the distance from j to i.

Complexity. For example, the increasingly popular Strong Exponential Time Hypothesis [IPZ01] states that a similar type of improved algorithm does not exist for CNF-SAT. Third, improved algorithms for TSP would show how to break the natural mold of DP over subsets from [Bel62, HK62], because within this mold it seems unavoidable to discard any of the $\binom{n}{n/2} \approx 2^n$ candidates of the $n/2$ first cities the salesman may visit. This is probably the main motivation for asking for improvements as in Open Question 1: We expect that progress on them will imply new algorithmic paradigms.

Algebraic Algorithms. An important case of (A)TSP is the case where all weights are either 0 or ∞. Then the input can be represented with a (directed) graph in which we need to detect a Hamiltonian cycle. Thus, before attacking Open Question 1 we first need to detect Hamiltonian cycles faster than [Bel62, HK62]. Fortunately, in the last decade a beautiful line of *algebraic algorithms* was developed that led to spectacular progress and culminated in a breakthrough randomized algorithm by Björklund [Bjö14] (see also [BHKK17]) that detects Hamiltonian cyles in undirected graphs in $O(1.66^n)$ time. See also a survey by Koutis and Williams [KW16a].

Inspired by [Bjö14], Cygan et al. [CNP+11] showed such an algebraic approach can be used to solve the Hamiltonicity problem fast on graphs of small 'treewidth'. A follow-up work [CKN18] introduced the *matchings connectivity matrix* (see Section 2.2 for a definition), determined its rank, and used it to detect Hamiltonian cycles in graphs of small 'pathwidth' and bipartite directed graphs.

On the positive side, these algorithms step out of the 'DP over subsets' mold of [Bel62, HK62] significantly. Especially the approach from [CNP+11, CKN18] is extremely flexible in how the solution is divided into sub-solutions, which make it the right approach in the case that the input graph has nice structure such as low treewidth.

However, these algorithms are algebraic, which means they count certain candidate solutions and use the power of algebraic cancellation (i.e. subtraction or cancellation modulo 2) to filter out the actual solutions. Extending such algorithm to weighted problems such as TSP can be done by counting candidates of fixed weight, but this typically incurs *pseudo-polynomial* time overhead in the runtime. Avoiding such overhead is a significant research challenge that, for example, also underlies the well-known open problem of solving the All-Pairs-Shortest-Path problem in sub-cubic time (see e.g. [Wil18]).

Rank-Based Method for TSP. For long, improvements over [Bel62, HK62] for TSP were only known in graphs of bounded (average) degree[2] based on branching or truncating the 'DP over subsets'-approach [BHKK12, CP15, Epp07, Geb08, IN07]. But in 2015, Bodlaender et al. [BCKN15] showed how to employ insights from the algebraic algorithms to weighted problems such as TSP while avoiding the aforementioned pseudo-polynomial overhead in the maximum input weight. They presented fast algorithms for TSP on instances in graphs of small treewidth that are very flexible in how sub-solutions are built similar to the tools from [CNP+11, CKN18].

The main observation from [BCKN15] was that, in any dynamic programming algorithm, the number of partial solutions could potentially be reduced significantly by using Gaussian elimination, if an appropriate 'partial solutions matrix' has sufficiently low rank. For (weighted) Hamiltonian cycle this turned out an especially useful approach since the associated partial solutions matrix is the aforementioned matchings connectivity matrix which has remarkably low rank [CKN18].

Given the rank-based method of [BCKN15] and the rank bound from [CKN18], it is natural to be optimistic about a resolution of Open Question 1. While there still turn out to be several obstacles, we suggest in this work that this optimism may be justified.

1.1 Our Contribution

For our main result, we restrict Open Question 1 to undirected bipartite graphs. The motivation for this is that (1) general directed graphs seem out of reach as even in the unweighted case a $O((2-\varepsilon)^n)$ time algorithm for $\varepsilon > 0$ remains elusive (see Subsection 1.2), and (2) the case of bipartite graphs was an important special case that also played a crucial role in the algorithm by Björklund [Bjö14].

Theorem 1. *Suppose that $(s \times s)$-matrices can be multiplied in $s^{2+o(1)}$ time. Then there is a Monte Carlo algorithm that, given an undirected bipartite graph $G = (L \cup R, E)$ and weights $w : E \to \mathbb{R}$, finds a Hamiltonian tour of minimum weight in $O(1.9999^n)$ time.*

All arithmetic operations on the input weights used in the algorithm behind Theorem 1 are additions and comparisons, thus in the commonly used computational random access model the run time is independent of the involved weights.

We let ω denote the smallest number such that $(s \times s)$-matrices can be multiplied in $s^{\omega+o(1)}$ time. Thus we assume $\omega = 2$. As the tiny slack in our run-times suggests, there exists an $\varepsilon > 0$ such that $\omega < 2+\epsilon$ implies a $O((2-\varepsilon)^n)$. We briefly further discuss the $\omega = 2$ assumption below.

Theorem 1 should be considered to be interesting independent of whether $\omega = 2$: It is (to the best of the author's knowledge) the first result that 'breaks a barrier' (e.g. improves over a relative naïve algorithm with a natural run-time) conditioned on whether $\omega = 2$. Thus, the fact that we are able to find improved algorithms even with this assumption indicates substantial progress on the open problem.

Minimum Weight Hamiltonian Pair. En route to Theorem 1, we significantly break the 'DP over subsets'-mold and decompose the sought tour in path systems (e.g. a collection of disjoint paths) similarly to [BCKN15, CKN18]. To be able to detect the pair of path-systems that jointly form an optimal tour, we need a fast algorithm for the following (to our knowledge, new) computational problem:

MinHamPair
Input: Families $\mathcal{A}, \mathcal{B} \subseteq \Pi_m([t]) \times \mathbb{R}$ of weighted perfect matchings
Asked: $(A, w_1) \in \mathcal{A}$ and $(B, w_2) \in \mathcal{B}$ with minimum w_1+w_2 and $A \cup B$ forming an Hamiltonian Cycle

One of our main technical contributions is a fast algorithm for this problem. Specifically:

Theorem 2. *There is a Monte Carlo algorithm for MinHamPair that solves instance $\mathcal{A}, \mathcal{B} \subseteq \Pi_m([t]) \times \mathbb{R}$ in $O((|\mathcal{A}| + |\mathcal{B}|)2^{3t/10} + 3^{t/2})$ time.*

[2] With every (A)TSP instance we can associate a (directed) graph formed by all (directed) arcs with finite weight.

The algorithm behind Theorem 2 relies on new structural insights on the *Matchings Connectivity Matrix*. For even $t \geq 2$ this a Boolean matrix $\mathbf{H}_t$ indexed by perfect matchings on K_t that indicates whether the two matchings form a Hamiltonian cycle. After a relatively simple preprocessing step to remove the weights, the algorithm behind Theorem 2 follows a variant of Freivalds' matrix multiplication verification algorithm (stated in Lemma 2.2) to check whether the matrix $\mathbf{H}_t[\mathcal{A}, \mathcal{B}]$ is non-zero using a factorization of $\mathbf{H}_t$.

Our crucial insight comes in here: A new factorization of $\mathbf{H}_t$, that we call the *narrow cut* factorization, can be used to check $\mathbf{H}_t[\mathcal{A}, \mathcal{B}]$ faster than possible with previous factorizations. Two of such factorizations were previously known [BCKN15, CKN18]:

- A *cut-based* factorization with inner dimension 2^{t-1} and relatively *sparse* factorizing matrices,
- A *matching-based* factorization with inner dimension $2^{t/2-1}$ and relatively *dense* factorizing matrices.

See Subsection 2.2 for more details. Curiously, using either factorization would lead to an $O((|\mathcal{A}| + |\mathcal{B}|)2^{t/2})$ time algorithm for MinHamPair for different reasons: One factorization is sparse and the other is narrow. Our new factorization is (relatively) narrow and sparse and therefore leads to an improved algorithm. The same factorization can also be used to obtain new simple algorithms for undirected Hamiltonicity and Directed Bipartite Hamiltonicity. We refer to Section 3 for more intuition and details.

Reducing Bipartite TSP to MinHamPair. To obtain Theorem 1 from Theorem 2 we use a slight variant of a known algorithm using the *rank-based method* by Bodlaender et al. [BCKN15, Theorem 3.9]. This algorithm solves TSP in $n(2+2^{\omega/2})^{\mathrm{pw}}\mathrm{pw}^{O(1)}$ time, if a path decomposition of the underlying graph G of pathwidth pw is given.[3] It does so by running a fairly straightforward dynamic programming algorithm that (roughly speaking) stores for every matching on a separator the minimum weight of a partial solution (i.e. a set of paths) that connects the vertices on the separator as dictated by the matching. Since there are $k^{O(k)}$ matchings on k vertices and a separator can be as large pw this implies a $\mathrm{pw}^{O(\mathrm{pw})}n$ time algorithm. To obtain the $2^{O(\mathrm{pw})}n$ run time, the rank-based method uses a `reducematchings` subroutine that employs Gaussian elimination to reduce the number of matchings to only $2^{O(\mathrm{pw})}$ that are representative for all matchings (see Definition 2.11). An important fact to note is that this Gaussian elimination step is a major bottleneck for the algorithm. If there would be a linear time implementation of `reducematchings`, the algorithm would run in $n(2+\sqrt{2})^{\mathrm{pw}}\mathrm{pw}^{O(1)}$ time (we continue this discussion below). See Section 2.3 for formal definitions.

Note that if the underlying graph G is bipartite with parts L, R, then pw can be assumed to be at most $n/2$. If $R = \{r_1, \ldots, r_{n/2}\}$ an easy path decomposition is obtained by adding L to each bag and iteratively introducing and forgetting r_i for $i = 1, \ldots, n/2$. If $\omega = 2$ the algorithm from [BCKN15] would already run in $2^n\mathrm{poly}(n)$ time, and the slow Gaussian elimination step is the only obstacle preventing further improvement.

[3] Though this paragraph relies on basic knowledge of path decompositions, it will not be used in the formal description.

To turn this into an $O(1.9999^n)$ time reduction, we (conceptually) split the sought tour T in $T_1 = T \cap E_1$ and $T_2 = E \cap E_2$ where $E_1 = E(G) \cap (L \times R_1)$ and $E_2 = E(G) \times (R \setminus R_1)$, $R_1 = r_1, \ldots, r_{n/2}$. We compute representative sets for all possibilities of T_1 and T_2. However, we settle for a slightly larger representative set than guaranteed by only applying `reducematchings` sporadically, thereby avoiding using $\Omega(2^n)$ time to compute the representative sets. Afterwards we use Theorem 2 to find the tour $T_1 \cup T_2$ of small weight.

Computing Representative Sets. As mentioned above, a bottleneck in the rank-based method is procedure `reducematchings` that computes a representative set of matchings. An ambitious open question is whether the run time of this algorithm can be improved to input-linear time. This would imply an $O((2+\sqrt{2})^{n/2})$ time algorithm for bipartite TSP by the algorithm from [BCKN15]. We do not make progress on this question, but present consequences of a relaxed version of this open question:

Theorem 3. *Let $\alpha, \beta, \varepsilon$ be such that $\alpha < 0.7 - \varepsilon$ and $\alpha + \beta < \log_2(3)/2 - \varepsilon$. Suppose there an algorithm that, given an n-vertex undirected graph G with edge weights $w : E(G) \to \mathbb{R}$ and a family of perfect matchings $\mathcal{A} \subseteq \Pi_m(V(G))$, computes a set $\mathcal{A}' \subseteq \mathcal{A}$ that represents $\mathcal{A}$ and satisfies $|\mathcal{A}'| \leq 2^{\alpha n}$ in $|\mathcal{A}|2^{\beta n}$ time. Then TSP can be solved in $O(2^{(1-\varepsilon')n})$ for some constant $\varepsilon' > 0$ that depends on $\varepsilon > 0$.*

The proof relies on a non-trivial randomized procedure to build representative sets of large subgraphs from small subgraphs. Afterwards, it applies Theorem 2 similarly as in the proof of Theorem 1. We believe that the structure of perfect matchings guaranteed in this special case makes Theorem 3 a promising route towards resolving the symmetric case of Open Question 1.

The Quadratic Matrix Multiplication Assumption. The current record upper bound for ω is $\omega < 2.3728639$ by Le Gall [Gal14], and it is popularly conjectured that $\omega = 2$. While conjectures even stronger than $\omega = 2$ have been made [CKSU05, Str94], much of the recent work focuses on limitations of the currently known methods [Alm19].

As mentioned before, our work is one of the first works that break a natural barrier assuming $\omega = 2$, and it should therefore be taken as an indication that a similar unconditional result is also within reach.

By using standard proof systems from Linear Algebra [DLP17] our methods also imply a 'Merlin-Arthur' proof (see e.g. [Wil16] for a definition) with $O(1.9999^n)$ proof size and verification time unconditionally. We briefly elaborate on this in Appendix A This is also interesting because the methods to obtain such fast protocols (see e.g. [Wil16]) are also 'algebraic' and do not seem able to deal with weighted variants.

1.2 Related Work

Algebraic Algorithms. The earliest algebraic algorithms for Hamiltonicity (and TSP) are by Karp [Kar82] and Gottlieb et al. [KGK77].

The aforementioned research line of 'algebraic algorithms' started with the work by Koutis [Kou08] and was at first used to obtain fast Fixed Parameter Tractable algorithms to find paths of length at least k. The currently fastest algorithms to detect paths on k

vertices are $1.66^k n^{O(1)}$ time in undirected graphs [BHKK17] and $2^k n^{O(1)}$ time in directed graphs [KW16b, Wil09].

Especially tantalizing is the current progress on detecting Hamiltonian Cycles in *directed* graphs: If d is a constant, the input graph has at most d^n Hamiltonian Cycles, their exact quantity can be determined in $O((2-c_d)^n)$ time for some constant $c_d > 0$ depending on d. Nevertheless, it remains an open problem whether directed Hamiltonian cycles can be detected in $O((2-\varepsilon)^n)$ time for some constant $\varepsilon > 0$ [BKK17].

All algorithms mentioned above are randomized. Improved deterministic algorithm are unknown even for bipartite graphs. As mentioned before, all these algorithm have consequences for the weighted (TSP) variant as well. For example, the algorithm [Bjö14] Björklund solves (undirected) TSP in $O(1.66^n W)$ time if all weights are in $\{1,\ldots,W\}$, and consequently it can also compute a $(1-\varepsilon)$-approximate optimal tour in $O(1.66/\varepsilon)$ time via standard rounding arguments.

The natural idea to use of sparse or narrow matrix factorizations to solve the detecting of a pair satisfying a certain relation based on Freivalds has the same general blueprint of some algorithms in quite different settings [BHKK09, CKN18, FLPS16].

Matchings Connectivity Matrix. The matchings connectivity matrix was defined in [CKN18], where the authors showed its rank over $\mathbb{F}_2$ equals $2^{t/2-1}$. They used their factorization in a way similar to how we use it in Subsection 3.3 to decompose the sought Hamiltonian cycle in two perfect matchings. Additionally they showed Hamiltonian cycles in graphs with given path decomposition of width pw can be detected in $(2+\sqrt{2})^{\mathrm{pw}} n^{O(1)}$ time and this cannot be improved to $(2+\sqrt{2}-\varepsilon)^{\mathrm{pw}} n^{O(1)}$ time for $\varepsilon > 0$ unless the Strong Exponential Time Hypothesis (SETH) fails.

The submatrix of the matchings connectivity matrix induced by all matchings on the complete bipartite graph was studied in [RS95]. They showed its rank over the reals is 2^t (up to $t^{O(1)}$ factors) using representation theory of the symmetric group. In [CLN18] the authors used representation theory of the symmetric group and various other tools from algebraic combinatorics to prove that the rank of $\mathbf{H}_t$ over the reals is 4^n, (up to $t^{O(1)}$ factors). They used this to show that Hamiltonian cycles cannot be counted in $(6-\varepsilon)^{\mathrm{pw}} n^{O(1)}$ time for any $\varepsilon > 0$ unless the SETH fails. Assuming $\omega = 2$, a matching upper bound exists as well [Wlo19].

Exponential Time Algorithms for TSP.. There have been many studies of exact algorithms for ATSP. A general result is that for every integer d there exists a constant $c_d > 0$ such that ATSP on graphs of average degree at most d can be solved in $O((2-c_d)^n)$ time [CP15] An $n^{1.5}2^n$ time algo was reported in an invited talk by Chan [Cha13]. Using quantum algorithms, ATSP can be solved in $O(1.729^n)$ time [ABI+19].

An especially big effort has been made on the space-time tradeoffs for ATSP. The best polynomial space algorithms run in $4^n n^{O(\log n)}$ time [GS87] and $2^n W$ time [LN10], where W denotes the maximum input weight.

1.3 Organization

The remainder of this paper is organized as follows: In Section 2 we outline the required preliminary knowledge and notation, in Section 3 we present the narrow cut factorization, which we subsequently use in Section 4 to prove Theorem 2. Theorem 2 is subsequently used in Section 5 to prove Theorem 1 and Theorem 3.

2 PRELIMINARIES

Given an integer t we use $[t]$ to denote $\{1,\ldots,t\}$. If t is Boolean, $[t]$ denotes 1 if t = **true** and 0 otherwise. In this paper $\equiv$ denotes congruence modulo 2. We use $\bar{a}$ to denote the binary complement of a bit-string a, and let ε denote the empty string.

If G is an undirected graph, $\Pi_{\mathrm{m}}(G)$ denotes the set of all perfect matchings of G. For a set U, we let $\Pi_{\mathrm{m}}(U)$ denote the set of all perfect matchings of the complete graph with vertex set U. We use $\Pi_2([t])$ for the family of subsets of all $[t]$ that contain the element 1. We let $\Pi(U)$ denotes the set of all partitions of U. This set gives rise to a lattice when partially ordered by coarsening with $\{U\}$ being the maximum element and the partition into singletons being the minimum element. We let $\sqcap$ and $\sqcup$ denote the meet and $\sqcup$ operations of this lattice. If A is a set we let $a \in_R A$ denote that a is a uniformly sampled element from A.

2.1 Matrices

All matrices in this paper are denoted with bold-face characters. We let $\mathbf{A} \in \mathbb{F}^{R\times C}$ denote that $\mathbf{A}$ is a matrix with rows R, columns C and elements from a field $\mathbb{F}$. If $X \subseteq R, Y \subseteq C$ we denote $\mathbf{A}[X,Y]$ for the submatrix of $\mathbf{A}$ induced by rows X and columns Y. To indicate no restriction we use $\cdot$ as an alternative, i.e. $\mathbf{A}[X,\cdot]$ and $\mathbf{A}[\cdot,Y]$ denote the matrix induced by all rows X, and respectively, columns Y. We use $\mathbf{A}^T$ to denote the transpose of $\mathbf{A}$. We let ω denote the smallest number such that two $(s\times s)$-matrices can be multiplied in $s^{\omega+o(1)}$ time. The current best bounds are the trivial lower bound $\omega \geq 2$, and the upper bound $\omega < 2.3728639$ by Le Gall [Gal14]. Given a matrix $\mathbf{A} \in \mathbb{F}^{R\times C}$, the *$k'$th Kronecker power* $\mathbf{A}^{\otimes k}$ is the matrix indexed with rows by R^k and columns by C^k such that

$$\mathbf{A}^{\otimes k}[r_1,\ldots,r_k,c_1,\ldots,c_k] = \prod_{i=1}^{k} \mathbf{A}[r_i,c_i].$$

Kronecker powers will be useful for us by virtue of the following lemma:

Lemma 2.1 (Yates' Algorithm [Yat37]). *Let $\mathbf{A} \in \mathbb{F}^{R\times C}$, k be an integer and $v \in \mathbb{F}^{R^k}$ given as input. Then $\mathbf{A}^{\otimes k}v$ can be computed in $O(\max\{|R|^{k+2},|C|^{k+2}\})$ time.*

The lemma is proved by a simple Fast-Fourier-Transform style procedure. We recommend [Kas18, Section 3.1] for a proof in modern language.

Lemma 2.2 ([Fre77]). *Let $\mathbb{F}$ be a field and $\mathbf{A} \in \mathbb{F}^{n\times n}$ be given as input. If $v \in_R \mathbb{F}^n$, then the probability that $\mathbf{A}v$ equals the all-zero vector is at most $\frac{1}{2}$.*

Using the random subsum principle (i.e. for every non-zero $w \in \mathbb{F}^d$, the inner product $\sum_{i=1}^{d} r_i w_i$ with a random vector $r \in_R \mathbb{F}^d$ is non-zero with probability at least 1/4) we obtain the following easy consequence:

Lemma 2.3. *Let $\mathbb{F}$ be a field and $\mathbf{A} \in \mathbb{F}^{n\times n}$. If $u,v \in_R \mathbb{F}^n$, then the probability that $u^T\mathbf{A}v$ equals the all-zero vector is at most $\frac{1}{4}$.*

2.2 The Matchings Connectivity Matrix and Its Factorizations

The following matrix will play a crucial role in this paper:

Definition 2.4 (Matchings Connectivity matrix). *For even $t \geq 2$, define $\mathbf{H}_t \in \{0,1\}^{\Pi_m([t])\times\Pi_m([t])}$ as*

$$\mathbf{H}_t[A,B] = \begin{cases} 1, & \text{if } A \cup B \text{ is an Hamiltonian Cycle,} \\ 0, & \text{otherwise.} \end{cases}$$

Definition 2.5 (Split matrix). *A cut $C \in \Pi_2([t])$ is* split *by a matching $A \in \Pi_m([t])$ if every edge of A is either contained in C or is disjoint from C. For even $t \geq 2$, define $\mathbf{S}_t \in \{0,1\}^{\Pi_m([t])\times\Pi_2([t])}$ as*

$$\mathbf{S}_t[A,C] = \begin{cases} 1, & \text{if } A \text{ is split by } C, \\ 0, & \text{otherwise.} \end{cases}$$

For a perfect matching $M \in \Pi_m([t])$ define function $\alpha_M\colon [t] \to [t]$ with $\alpha_M(i) = j$ if and only if $\{i,j\} \in M$, i.e., α_M maps each element of U to its partner in the perfect matching M.

Definition 2.6 (Basis Matchings from [CKN18]). *Let $X(\varepsilon) := \{\{1,2\}\}$ and $\mathcal{X}_2 := \{X(\varepsilon)\}$. Let $t \geq 4$ be an even integer and let $a \in \{0,1\}^{t/2-2}$. Define perfect matchings $X(a0)$ and $X(a1)$ on $[t]$ as follows, using $\alpha := \alpha_{X(a)}$:*

$$X(a1) := X(a) \cup \{\{t-1,t\}\},$$

$$X(a0) := (X(a) \setminus \{\{t-2,\alpha(t-2)\}\}) \cup \{\{t-2,t\},\{t-1,\alpha(t-2)\}\}.$$

Finally, we let $\mathcal{X}_t$ be the family of perfect matchings $X(a)$ for any $a \in \{0,1\}^{t/2-1}$.

The matching $X(a)$ can be pictorially constructed from vector $a \in \{0,1\}^{t/2-1}$ as follows: Draw the integers $1,\ldots,t$ and draw a vertical 'bar' before each even integer. For $i \leq t-1$ associate the value a_i with the i'th bar placed between $2i-1$ and $2i$. To form the matching $X(a)$ we iterate over $i = 1,\ldots,t/2$ and connect the yet unmatched vertex from $\{2i-2, 2i-1\}$ with $2i$ if $a_i = 1$ and with $2i+1$ if $a_i = 0$. See Figure 1 for an illustration.

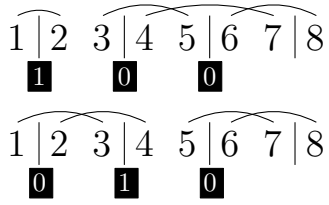

Figure 1: The basis matchings $X(100)$ and $X(010)$.

The following factorization shows that $\mathbf{H}_t$ has rank only $2^{t/2-1}$ over $\mathbb{F}_2$.

Lemma 2.7 (Matchings Factorization [CKN18]). *If $A, B \in \Pi_m([t])$, then*

$$\mathbf{H}_t[A,B] \equiv \sum_{a\in\{0,1\}^{t/2-1}} \mathbf{H}_t[A,X(a)] \cdot \mathbf{H}_t[X(\bar{a}),B].$$

We often prefer to write formulas as the above in matrix language to keep notation clean. To do so, we use the following facilitating matrix:

Definition 2.8 (Flip Matrix). *Let $\mathbf{F}_t \in \{0,1\}^{t/2-1}\times\{0,1\}^{t/2-1}$ denote the matrix that satisfies $\mathbf{F}_t[a,b] = 1$ if $b = \bar{a}$ and $\mathbf{F}_t[a,b] = 0$ otherwise.*

With the definition of $\mathbf{F}_t$ in hand, Lemma 2.7 can be written as the following matrix multiplication:

$$\mathbf{H}_t \equiv \mathbf{H}_t[\cdot,\mathcal{X}_t] \cdot \mathbf{F}_t \cdot \mathbf{H}_t[\mathcal{X}_t,\cdot] \tag{1}$$

We will also need the following easier, but less narrow, factorization based on cuts. It was implicitly used in [CNP+11] and made more explicit in [BCKN15].

Lemma 2.9 (Cut Factorization [BCKN15]). $\mathbf{H}_t \equiv_2 \mathbf{S}_t\mathbf{S}_t^T$. *More explicitly, if $A, B \in \Pi_m([t])$, then*

$$\mathbf{H}_t[A,B] \equiv \sum_{C\in\Pi_2([t])} \mathbf{S}_t[A,C] \cdot \mathbf{S}_t^T[B,C].$$

2.3 Representative Sets via Gaussian Elimination

We list preliminary key properties of representative sets as formalized in [BCKN15]. These are listed here to facilitate the high level overview of the approach in Section 5.

Definition 2.10. *A* set of weighted partitions *is a set $\mathcal{A} \subseteq \Pi(U)\times\mathbb{N}$, i.e., a family of pairs, each consisting of a matching in $\Pi_m([t])$ and a non-negative integer weight.*

Definition 2.11 (Representation). *Given a set of weighted partitions $\mathcal{A} \subseteq \Pi(U)\times\mathbb{N}$ and a partition $q \in \Pi(U)$, define*

$$\mathtt{opt}(q,\mathcal{A}) = \min\{w \mid (p,w) \in \mathcal{A} \wedge p \sqcup q = \{U\}\}.$$

For another set of weighted partitions $\mathcal{A}' \subseteq \Pi(U)\times\mathbb{N}$, we say that $\mathcal{A}'$ represents *$\mathcal{A}$ if for all $q \in \Pi(U)$ it holds that $\mathtt{opt}(q,\mathcal{A}') = \mathtt{opt}(q,\mathcal{A})$.*

Theorem 4 (Corollary 3.17 in [BCKN15]). *There exists an algorithm* `reducematchings` *that given set of weighted matchings $\mathcal{A} \subseteq \Pi_m(U)\times\mathbb{N}$, outputs in $|\mathcal{A}|2^{\frac{(\omega-1)}{2}|U|}|U|^{O(1)}$ time a set of weighted matchings $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{A}'$ represents $\mathcal{A}$ and $|\mathcal{A}'| \leq 2^{|U|/2}$, where ω denotes the matrix multiplication exponent.*

3 NARROW CUT FACTORIZATION

In this section we present the narrow cut factorization. As mentioned in the introduction, it is obtained as combination of the cut-factorization (Lemma 2.9 and matching factorization (Lemma 2.7).

In Subsection 3.1 we describe the family of cuts that will index the inner-dimension of the factorization, along with some useful properties. In Subsection 3.2 we will use these to state and proof the factorization. While the proof of Theorem 2 in the next section will crucially rely on this factorization, we also provide another application of the factorization to reobtain some known results in a different way in Subsection 3.3.

3.1 Narrow Cut Basis

We now define the inner dimension of the factorization that we refer to as the narrow cut basis. It's definition has the similar structure as the basis matchings from Definition 2.6. This is no coincidence: Intuitively, the narrow cut basis can be defined as all cuts that split some basis matching.

Definition 3.1 (Narrow Cut Basis). *Let $C(\varepsilon) := \{1,2\}$ and let $C_2 = \{\{1,2\}\}$ Let $t \geq 4$ be even and let $a \in \{0,1,2\}^{t/2-2}$. Define subsets $C(a0)$, $C(a1)$ and $C(a2)$ of $[t]$ as follows:*

$$C(a0) := \begin{cases} C(a), & \text{if } t-2 \in C(a) \\ C(a) \cup \{t-1,t\}, & \text{otherwise.} \end{cases}$$

$$C(a1) := \begin{cases} C(a) \setminus \{t-2\} \cup \{t-1\}, & \text{if } t-2 \in C(a) \\ C(a) \cup \{t-2,t\}, & \text{otherwise.} \end{cases}$$

$$C(a2) := \begin{cases} C(a) \cup \{t-1,t\}, & \text{if } t-2 \in C(a) \\ C(a), & \text{otherwise.} \end{cases}$$

Finally, let $\mathcal{C}_t$ be the family of subsets $C(a)$ for any $a \in \{0,1,2\}^{t/2-1}$.

The cut $C(a)$ can be pictorially constructed from vector $a \in \{0,1,2\}^{t/2-1}$ in the following way that is analogous to the construction of $X(a)$ in Subsection 2.2: Draw the integers $1,\ldots,t$ and draw a vertical 'bar' before each even integer. Start with the cut $\{1,2\}$ and for $i = 2,\ldots,t/2-1$, read the 'state' (i.e. is a vertex in the current cut) of $2i$ and if

- $a_i = 0$, let $2i+1$ and $2i+2$ have the opposite state of $2i$,
- $a_i = 1$, let $2i+1$ have the same state as $2i$, $2i+2$ the opposite state of $2i$, and flip the state of $2i$,
- $a_i = 2$ copy the state of $2i$ to $2i+1$, $2i+2$.

See also Figure 2 for two examples. Note that $1 \in C$ for every $C \in \mathcal{C}_t$ so indeed $\mathcal{C}$ is a family of cuts.

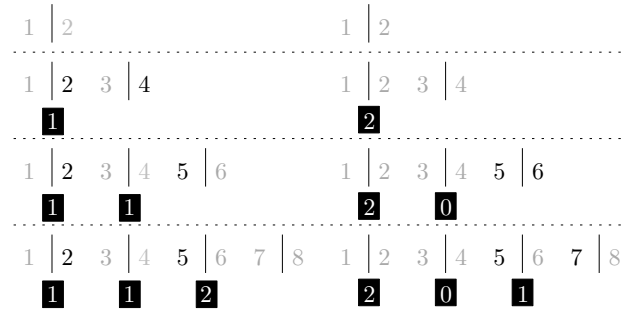

Figure 2: The basis cuts $C(1), C(11), C(112)$ on the left and $C(2), C(20), C(201)$ on the right.

Lemma 3.2. *Let $t \geq 2$, $a \in \{0,1,2\}^{t/2-1}$ and $b \in \{0,1\}^{t/2-1}$. Then $C(a)$ splits $X(b)$ if and only if $a_i \neq b_i$ for every $i = 1,\ldots,t/2-1$.*

Proof. We use induction on t. For $t = 2$ the statement clearly holds since $C(\varepsilon)$ splits $X(\varepsilon)$, so assume $t \geq 4$. Let $x \in \{0,1,2\}$ and $y \in \{0,1\}$. We will use a case analysis to show that $C(ax)$ splits $X(by)$ if and only if $C(a)$ splits $X(b)$ and $x \neq y$.

Case 1: $t-2 \in C(a)$. We distinguish cases depending on x:

- **$C(a0)$** contains $t-2$ but not $t-1$ and t.
  - **$X(b0)$** contains the edge $\{t-2,t\}$ and will therefore not be split by $C(a0)$.
  - **$X(b1)$** contains the edge $\{t-1,t\}$ that is split by $C(a0)$. Since $C(a0) \cap [t-2] = C(a)$, the remaining edges of $X(b1)$ are split by $C(a0)$ if and only if $C(a)$ splits $X(b)$.
- **$C(a1)$** contains $t-1$ but not $t-2$ and t.
  - **$X(b1)$** contains the edge $\{t-1,t\}$ and will therefore not be split by $C(a1)$.
  - **$X(b0)$** contains the edge $\{t-2,t\}$ that is split by $C(a1)$. Recall $\alpha_{X(b0)}(t-1) = \alpha_{X(b)}(t-2)$. Since $t-2 \in C(a)$ and $t-1 \in C(a1)$, the remaining edges of $X(b0)$ are split by $C(a1)$ if and only if $C(a)$ splits $X(b)$.
- **$C(a2)$** contains $t-2, t-1$ and t and therefore splits the edge of $X(by)$ incident to t. Since $C(a2) \cap [t-2] = C(a)$ the remaining edges are split by $C(a2)$ if and only if $C(a)$ splits $X(b)$.

Case 2: $t-2 \notin C(a)$. We again distinguish cases depending on x:

- **$C(a0)$** contains $t-1$ and t but not $t-2$.
  - **$X(b0)$** contains the edge $\{t-2,t\}$ and will therefore not be split by $C(a0)$.
  - **$X(b1)$** contains the edge $\{t-1,t\}$ that is split by $C(a0)$. Since $C(a0) \cap [t-2] = C(a)$, the remaining edges of $X(b1)$ are split by $C(a0)$ if and only if $C(a)$ splits $X(b)$.
- **$C(a1)$** contains $t-2$ and t, but not $t-1$.
  - **$X(b1)$** contains the edge $\{t-1,t\}$ and will therefore not be split by $C(a1)$.
  - **$X(b0)$** contains the edge $\{t-2,t\}$ that is split by $C(a1)$. Recall $\alpha_{X(b0)}(t-1) = \alpha_{X(b)}(t-2)$. Since $t-2 \notin C(a)$ and $t-1 \notin C(a1)$, the remaining edges of $X(b0)$ are split by $C(a1)$ if and only if $C(a)$ splits $X(b)$.
- **$C(a2)$** does not contain $t-2, t-1$ and t and therefore splits the edge of $X(by)$ incident to t. Since $C(a2) \cap [t-2] = C(a)$ the remaining edges are split by $C(a2)$ if and only if $C(a)$ splits $X(b)$.

□

The following property that was already mentioned in the intuition above makes the basis cuts so useful:

Lemma 3.3. *Let $X \in \mathcal{X}_t$ be a basis matching and $C \in \Pi_2([t])$ be a cut that splits X. Then $C \in \mathcal{C}_t$.*

Proof. Since C splits X it contains $\{1, \alpha_X(1)\}$, and either contains or avoids all the other $t/2-1$ edges of X. Thus there are $2^{t/2-1}$ cuts $C \in \Pi_2([t])$ that split M. Thus, to prove the lemma, it suffices to show there also exist $2^{t/2-1}$ cuts $C \in \mathcal{C}_t$ that split M. This follows directly from Lemma 3.2: If $X = X(b)$ for $b \in \{0,1\}^{t/2-1}$, there are $2^{t/2-1}$ possibilities for $a \in \{0,1,2\}^{t/2-1}$ with $a_i \neq b_i$ for every i. □

3.2 The Factorization

We are now ready to present the main result of this section.

Lemma 3.4 (Narrow Cut Factorization). *Let $t \geq 2$ be an even integer. Unify $\mathcal{C}_t$ and $\{0,1,2\}^{t/2-1}$ with the bijection $a \mapsto C(a)$. Then there exists a (3×3)-matrix $\mathbf{B} \in \{0,1\}^{\{0,1,2\} \times \{0,1,2\}}$ such that*

$$\mathbf{H}_t \equiv \mathbf{S}_t[\cdot, \mathcal{C}_t] \cdot \mathbf{B}^{\otimes t/2-1} \cdot (\mathbf{S}_t[\cdot, \mathcal{C}_t])^T.$$

Proof. Let $A \in \Pi_{\mathrm{m}}([t])$ and $X \in \mathcal{X}_t$. By Lemma 2.9 we have

$$\mathbf{H}_t[A,X] \equiv \sum_{C \in \Pi_2([t])} \mathbf{S}_t[A,C]\mathbf{S}_t[X,C] = \sum_{C \in \mathcal{C}_t} \mathbf{S}_t[A,C]\mathbf{S}_t[X,C],$$

where the second equality uses that all summands indexed by cuts not in $\mathcal{C}$ vanish by Lemma 3.3. Thus, in matrix notation, we have

$\mathbf{H}_t[\cdot, \mathcal{X}_t] \equiv \mathbf{S}_t[\cdot, C_t] \cdot (\mathbf{S}_t[\mathcal{X}, C_t])^T$. Expanding in (1) gives:

$$\begin{aligned}\mathbf{H}_t &\equiv \left(\mathbf{S}_t[\cdot, C_t] \cdot (\mathbf{S}_t[\mathcal{X}, C_t])^T\right) \cdot \mathbf{F} \cdot \left(\mathbf{S}_t[\cdot, C_t] \cdot (\mathbf{S}_t[\mathcal{X}, C_t])^T\right)^T \\ &\equiv \mathbf{S}_t[\cdot, C_t] \cdot \left((\mathbf{S}_t[\mathcal{X}, C_t])^T \cdot \mathbf{F} \cdot \mathbf{S}_t[\mathcal{X}, C_t]\right) \cdot (\mathbf{S}_t[\cdot, C_t])^T,\end{aligned}$$

and it remains to rewrite the middle expression in parentheses into $\mathbf{B}^{\otimes t/2-1}$, for some matrix $\mathbf{B}$. By Lemma 3.2 we have that $\mathbf{S}_t[X(a), C(b)] = 1$ if and only if $a_i \neq b_i$ for every i. Let $p = t/2 - 1$ and index $\mathbf{S}_t[\mathcal{X}, C_t]$ with vectors in $\{0,1\}^{t/2-1} \times \{0,1,2\}^{t/2-1}$. Then we see that

$$\mathbf{S}_t[\mathcal{X}, C_t] = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}^{\otimes p},$$

and thus we have

$$\begin{aligned}\mathbf{B} &= (\mathbf{S}_t[\mathcal{X}, C_t])^T \cdot \mathbf{F} \cdot \mathbf{S}_t[\mathcal{X}, C_t] \\ &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}^{\otimes p} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^{\otimes p} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}^{\otimes p} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}^{\otimes p},\end{aligned}$$

and the lemma follows. □

3.3 Hamiltonicity via the Narrow Cut Factorization

In this subsection we demonstrate a simple application of the Narrow Cut Factorization for Hamiltonicity. These algorithmic results simplify and improve a similar approach from [CKN18], but are in turn inferior to the results from [Bjö14] and [BKK17]. We nevertheless present the result here since it already follows from a combination of Lemma 3.4 with standard methods.

Theorem 5. *There are randomized algorithms to detect Hamiltonian cycles in undirected graphs and directed bipartite graphs on n vertices in $O^*(3^{n/2})$ time.*

The proof relies on the following well-known definition and lemma:

Definition 3.5 (Tutte Matrix [Tut47]). *Let $G = (V, E)$ be a graph with linear ordering $\prec$ on V, let $\mathbb{F}$ be a field and for every $i < j$ let $x_{ij} \in \mathbb{F}$. Define*

$$A_G^{(x)}[i, j] = \begin{cases} x_{ij} & \text{if } \{i, j\} \in E \text{ and } i \prec j, \\ -x_{ji} & \text{if } \{i, j\} \in E \text{ and } j \prec i, \\ 0 & \text{otherwise}. \end{cases}$$

Lemma 3.6 ([Tut47]). *The determinant $\det(A_G^{(x)})$ is the polynomial in variables $x_{\{i,j\}}$ satisfying*

$$\det(A_G^{(x)}) = \sum_{M \in \Pi_m(G)} \prod_{\substack{\{i,j\} \in M \\ i<j}} x_{ij}^2.$$

Proof of Theorem 5. The algorithm is outlined in Algorithm 1. Note it takes $O^*(3^{n/2})$ time because there are $3^{n/2-1}$ iterations of the loop at Line 2, the determinants on Lines 3, 4 are computed in polynomial time with standard algorithms, and the product on Line 5 can be computed in $O^*(3^{n/2})$ using Yates' algorithm

Algorithm 1 Undirected Hamiltonicity via the Narrow Cut Factorization.

Algorithm `undirectedHamiltonicity`$(G = (V, E))$
Output: **true** with probability at least $1/2$ if G is Hamiltonian, and **false** otherwise.
1: For each $\{i, j\} \in E$ with $i < j$ pick $x_{ij}, y_{ij} \in_R GF(2^k)$, where $k = \log_2 4n$
2: **for** $a \in \{0,1,2\}^{n/2-1}$ **do**
3: $\quad l_i \leftarrow \det(A^{(x)}_{G[C(a)]}) \cdot \det(A^{(x)}_{G[V \setminus C(a)]})$
4: $\quad r_i \leftarrow \det(A^{(y)}_{G[C(a)]}) \cdot \det(A^{(y)}_{G[V \setminus C(a)]})$
5: $res \leftarrow l^T \cdot \mathbf{B}^{\otimes n/2-1} \cdot r$
6: **if** $res \neq 0$ **then return true else return false**

(Lemma 2.1). For correctness, notice the algorithm evaluates the polynomial $p(x, y) :=$

$$\begin{aligned}&\sum_{\substack{a,b \in \{0,1,2\}^{n/2-1} \\ \forall i: a_i \neq b_i}} \det\left(A^{(x)}_{G[C(a)]}\right) \det\left(A^{(x)}_{G[V \setminus C(a)]}\right) \cdot \\ &\qquad \det\left(A^{(y)}_{G[C(b)]}\right) \det\left(A^{(y)}_{G[V \setminus C(b)]}\right) \\ &= \sum_{\substack{a,b \in \{0,1,2\}^{n/2-1} \\ \forall i: a_i \neq b_i}} \left(\sum_{\substack{M_1 \in \Pi_m(G) \\ C(a) \text{ splits } M_1}} \prod_{\substack{\{i,j\} \in M_1 \\ i<j}} x_{ij}^2 \right) \cdot \\ &\qquad \left(\sum_{\substack{M_2 \in \Pi_m(G) \\ C(b) \text{ splits } M_2}} \prod_{\substack{\{i,j\} \in M_2 \\ i<j}} y_{ij}^2 \right) \\ &= \sum_{M_1, M_2 \in \Pi_m(G)} \left(\sum_{\substack{a,b \in \{0,1,2\}^{n/2-1} \\ \forall i: a_i \neq b_i}} [C(a) \text{ splits } M_1][C(b) \text{ splits } M_2] \right) \cdot \\ &\qquad \left(\prod_{\substack{\{i,j\} \in M_1 \\ i<j}} x_{ij}^2 \right) \left(\prod_{\substack{\{i,j\} \in M_2 \\ i<j}} y_{ij}^2 \right) \\ &\equiv \sum_{M_1, M_2 \in \Pi_m(G)} \mathbf{H}_t[M_1, M_2] \left(\prod_{\substack{\{i,j\} \in M_1 \\ i<j}} x_{ij}^2 \right) \left(\prod_{\substack{\{i,j\} \in M_2 \\ i<j}} y_{ij}^2 \right).\end{aligned}$$

Here the first equality follows from Lemma 3.6, and the second equality is a simple reordering of summations. The last equivalence (in the field $GF(2^k)$) uses the narrow cut factorization Lemma 3.4: in the third expression, the number of possibilities of a, b that lead to summand of value 1 will be 1 if $M_1 \cup M_2$ is an Hamiltonian cycle and 0 otherwise, and since $GF(2^k)$ has characteristic the equivalence follows.

Thus $p(x, y)$ is a polynomial of degree at most $2n$ with a separate monomial for each pair of perfect matchings that form a Hamiltonian cycle. Thus Algorithm 1 will never output **true** if G does not have a Hamiltonian cycle.

Moreover, the monomials will not cancel each other since a Hamiltonian cycle has a unique unordered decomposition into two perfect matchings, and the two different variables x, y ensure

they contribute distinct monomials. Thus p will not be the zero polynomial if G has a Hamiltonian cycle, and the Demillo-Lipton-Schwartz-Zippel polynomial identity testing lemma [DL78, Sch80] states that, since $2^k \geq 4n$, an evaluation of P at a random point will be non-zero with probability at least $1/2$.

For Bipartite Directed Hamiltonicity a very similar algorithm may be used. If $G = (L \cup R, E_1 \cup E_2)$ where $E_1 \subseteq L \times R$ and $E_2 \subseteq R \times L$, we count pairs of perfect matchings that are subsets of E_1 and E_2 respectively. □

4 THE HAMILTONIAN PAIR PROBLEM

The main contribution of this section a fast algorithm for the MinHamPair problem. For concenience, we first restate the result:

Theorem 2 (restated). *There is a randomized algorithm for MinHamPair that solves an instance $\mathcal{A}, \mathcal{B} \subseteq \Pi_m([t])$ in $O((|\mathcal{A}| + |\mathcal{B}|)2^{0.3t} + 3^{t/2})$ time.*

The algorithm behind Theorem 2 heavily builds on the narrow cut factorization, Lemma 3.4, from the previous section. The main technical effort is to obtain an algorithm with the same runtime as in Theorem 2 for the following special case of MinHamPair:

HamPair
Input: Two families $\mathcal{A}, \mathcal{B} \in \Pi([t])$
Asked: Whether there exists a pair $A \in \mathcal{A}$, $B \in \mathcal{B}$ such that $A \cup B$ is an Hamiltonian cycle.

We will discuss this in Subsection 4.1. Afterwards Theorem 2 is obtained by simple reduction from MinHamPair to HamPair outlined in Subsection 4.2.

4.1 Algorithm for the Unweighted Problem

In this subsection we present an algorithm to prove an unweighted analogue of Theorem 2:

Lemma 4.1. *There is a randomized algorithm for HamPair that solves an instance $\mathcal{A}, \mathcal{B} \subseteq \Pi_m([t])$ in $O((|\mathcal{A}| + |\mathcal{B}|)2^{0.3t} + 3^{t/2})$ time.*

The corresponding algorithm is listed in Algorithm 2. We start with describing the intuition and all intermediate lemmas before we proceed to the formal proof of Lemma 4.1. The intuition behind the algorithm is as follows: We first preprocess $\mathcal{A}$ and $\mathcal{B}$ to ensure each matching satisfies certain nice properties that we will explain later. Let $\mathcal{A}_2, \mathcal{B}_2$ be the resulting sets. We are interested in whether the matrix $\mathbf{H}_t[\mathcal{A}_2, \mathcal{B}_2]$ equals the all-zero matrix. By Lemma 2.3 we can check this by picking $u \in \mathbb{F}_2^{|\mathcal{A}|}$ and $v \in \mathbb{F}_2^{|\mathcal{B}|}$ and evaluating $u^T \mathbf{H}_t[\mathcal{A}_2, \mathcal{B}_2]$v. Algorithm 2 does this by using the narrow cut factorization and exploits that the cut-split-matrices are *sparse*.

To get the target run time, this means we need that the given matchings are on average consistent with at most $2^{0.3t}$, and we can enumerate them in this time bound. This enumeration will be done by the subroutine `enumCuts`(A) that will output the set of all basis cuts that split the matching A.

Unfortunately, the number of basis cuts that split A can be too large: For example if M is a basis matching it will be split by $2^{t/2-1}$ cuts by Lemma 3.2. But this seems a very special case: Any matching is split by $2^{t/2-1}$ cuts, and since there the probability that a random cut is a basis cut is exponentially small, we expect this number to be much lower is M is a *random matching*. To make this intuition work, we therefore apply a random permutation to all matchings as a preprocessing step.

More formally, we start with discussing how to enumerate basis cuts that split a given matching quickly. To do so we use preprocessing to ensure all matchings have small cutwidth:

Definition 4.2 (Cutwidth of Matching). *The* cutwidth *of M is defined as*

$$\mathrm{ctw}(M) = \max_{1 \leq j < n} \left|\left\{\{i,k\} \in M \,\middle|\, i \leq j \wedge j < k\right\}\right|.$$

That is, we think of the i'th *cut* as the set of edges with an coordinate at most i and a coordinate larger than i. For a random permutation, the expected number of edges split by the $t/4$'th cut is $n/2$ by linearity of expectation. Via standard arguments it can be shown that the cutwidth will deviate little from this with high probability:

Lemma 4.3 (Cutwidth of a Matching). *Let $M \in \Pi_m([t])$, π be a random permutation and let $M' := \pi(M)$. For large enough t, the probability that $\mathrm{ctw}(M') \geq 0.26t$ is at most $1/2^{\Omega(t)}$.*

Algorithm 2 Algorithm for the HamPair problem.

Algorithm `hamPair`$(\mathcal{A}, \mathcal{B})$ Assumes $\mathcal{A}, \mathcal{B} \subseteq \Pi_m([t])$
Output: **true** with probability $1/2$ if $\exists A, B \in \mathcal{A} \times \mathcal{B}$ such that $\mathbf{H}_t[A, B] = 1$, **false** otherwise.
1: Let $\pi : [t] \leftrightarrow [t]$ be a random permutation
2: Let $\mathcal{A}_1 := \{\pi(A) : A \in \mathcal{A}\}$, and $\mathcal{B}_1 := \{\pi(A) : A \in \mathcal{A}\}$
3: Let $\mathcal{A}_2, \mathcal{B}_2$ by removing matchings from $\mathcal{A}_2, \mathcal{B}_2$ of cutwidth at least $0.26t$
4: Let $\mathcal{A}_3, \mathcal{B}_3$ be obtained from $\mathcal{A}_2, \mathcal{B}_2$ by removing each element of $\mathcal{A}$ and $\mathcal{B}$ with probability $\frac{1}{2}$
5: **for** $A \in \mathcal{A}_3, C \in$ `enumCuts`(A) **do**
6: $\quad l_C \leftarrow l_C + 1 \mod 2$
7: **for** $B \in \mathcal{B}_3, C \in$ `enumCuts`(B) **do**
8: $\quad r_C \leftarrow r_C + 1 \mod 2$
9: $res \leftarrow l^T \cdot \mathbf{B}^{\otimes t/2-1} \cdot r$
10: **if** $res \neq 0$ **then return true else return false**

Proof. Consider the cut at position $i \leq t/2$, and let $0.26t \leq k \leq i$. The number of partitions of $[t]$ in subsets V_1, V_2 such that $|V_1| = i$ and k edges of M have both a vertex in V_1 and in V_2 is

$$\begin{aligned}
\binom{t/2}{k, (i-k)/2, (t-i-k)/2} 2^k &\leq k \binom{t/2}{k/2, k/2, (i-k)/2, (t-i-k)/2} \\
&= k \binom{t/2}{i/2} \binom{i/2}{k/2} \binom{t/2-i/2}{k/2} \\
&= k \binom{t/2}{i/2} \binom{t/2}{k/2} \\
&\leq k \binom{t/2}{i/2} \binom{t/2}{i/2} / 2^{\Omega(t)} \\
&\leq \binom{t}{i} / 2^{\Omega(t)}.
\end{aligned}$$

Thus with probability at least $1 - 1/2^{\Omega(t)}$, the number of edges at the i'th cut will be at most $0.26t$ for any i and the lemma follows by a union bound. □

We now focus on enumerating basis cuts that split a given matching of cutwidth at most $0.26t$. We first show these cuts can be enumerated in amortized polynomial time, if there are sufficiently many. Intuitively, the approach is to formulate a relatively straightforward dynamic programming table, and use that dynamic programming algorithms can be used to enumerate solution quickly once the table it built.

Lemma 4.4. *There is an algorithm* `enumCuts` *that, given* $M \in \Pi_m([t])$*, enumerates all cuts* $C \in \mathcal{C}_t$ *that split* M *in* $(2^{\mathrm{ctw}(M)} + |O|)t^{O(1)}$*, where* $|O|$ *denotes the output.*

Proof. Let $M_j := \{(i,k) \in M : i < j \wedge j < k\}$ to be all edges crossing the 'j'th cut'. For even $j = 2, 4, \ldots, n$ and $X \subseteq \{i \mid \{i,k\} \in M_j, i < k\}$ define $T_j^{\beta}[X]$ to be **true** if and only if there exists a bit string $a \in \{0,1,2\}^{j/2-2}$ such that

$$X = \left(\left\{ i \,\middle|\, \exists \{i,k\} \in M_j : i < k \right\} \cap C(a\beta) \right).$$

That is, there exists a 'partial' basic cut that has intersection X with the set of all endpoints of M_i. Using the recursive definition of a basis cut from Definition 3.1, it is easy to check that $T_j^{\beta}[X]$ satisfies a recursion of the type

$$T_j^{\beta}[X] = T_{j-2}^{0}[X_0] \vee T_{j-2}^{1}[X_1] \vee T_{j-2}^{2}[X_2],$$

where X_0, X_1, X_2 depend on the value of β and whether $\alpha_M(j-1)$ and $\alpha_M(j-2)$ are larger than j, at most j or either others neighbors.

Note the number of table entries of this dynamic programming algorithm is $2^{\mathrm{ctw}(M)}t^{O(1)}$. To see that we can enumerate all basis cuts consistent with M, note we are interested in $T_t^0[\emptyset], T_t^1[\emptyset], T_t^2[\emptyset]$ and if any of them are true can be backtrack in a straightforward manner to enumerate all basis cuts that make them true. The runtime overhead is only polynomial per basis cut since we enumerate a basis cut only once and in the algorithm only explores table entries that are **true** which ensures in each branch a basis cut will be found. □

To analyze the runtime of the Algorithm 2, it remains to bound the expected number of basis cuts consist with a matching. To this end, the following bound will be useful:

Lemma 4.5 (Number of basis cuts). *For every even* k *the number of basis cuts of cardinality* k *satisfies*

$$\left| \mathcal{C}_t \cap \binom{[t]}{k} \right| \leq \sum_{c=0}^{t/2} \binom{\frac{k}{2}}{c/2} \binom{\frac{t-k}{2}}{c/2} 2^c.$$

Proof. We show that a cut in $\mathcal{C}_t \cap \binom{[t]}{k}$ can be encoded in a unique way as a quadruple (c, A_e, A_h, A_f), where $c \leq t/2$ is an integer, $A_h \subseteq [c]$, $A_e \in \binom{k/2}{c/2}$ and $A_f \in \binom{(t-k)/2}{c/2}$.

Note this is sufficient to prove the desired claim as the number of such quadruples equals the claimed upper bound on the basis cuts.

Fix a basis cut $C(a)$ satisfying $|C(a)| = k$. For even $j < t$, we refer to $j, j+1$ as a group and call it *empty*, *half* and *full* if $|\{j, j+1\} \cap C(a)|$ equal $0, 1$ and 2, respectively. Similarly, we call group t full if $t \in C(a)$ and we call it empty otherwise.

The first parameter c in the encoding describes the number of half groups. This implies that there are $(k-c)/2$ full groups and thus $(t-k-c)/2$ remaining empty groups.

The crucial observation that directly follows from the definition of the basis cuts is that all full groups can only occur after an even number of half groups and all empty groups can only occur after an odd number of half groups.

Therefore, we can describe the state (i.e. empty, half or full) of each group with c and two integer partitions

$$e_1 + e_2 + \cdots + e_{c/2+1} = e \qquad f_1 + f_2 + \cdots + f_{c/2+1} = f.$$

into non-negative integers where $e = (t-k-c)/2$ is the total number of empty groups and $f = (k-c)/2$ is the total number of full groups. Note that for both the empty and full groups we have at most $c/2$ alternative on between two which two consecutive half groups we place them. For an illustration see Figure 3.

It is well known that the number of integer partitions $a_1 + \ldots + a_k = a$ into non-negative integers can be injectively encoded as a subset $A \subseteq \binom{a+k}{k}$. Thus we can encode $e_1, \ldots, e_{c/2+1}$ as A_e and $f_1, \ldots, a_{c/2+1}$ as A_f. This uniquely determines which is group is empty, half and full, and it only remains to describe of each half group which vertex is in and which one is not. For this, the remaining set A_e can be used. □

$$f\,f\;c\;e\,e\;c\;f\,f\,f\;c\;e\,c\,c\;e\,c\;f\;c\;e$$

Figure 3: An example of the encoding from the proof of Lemma 4.5 with $(f_1, f_2, f_3, f_4) = (2, 3, 0, 1)$ and $(e_1, e_2, e_3, e_4) = (2, 1, 1, 1)$.

Given Lemma 4.5, the step of taken a random permutation is now used as already intuitively described above:

Lemma 4.6. *Let* $M \in \Pi_m([t])$ *and let* $\pi : [t] \leftrightarrow [t]$ *be a random permutation. Then*

$$\mathbb{E}[|\{C \in \mathcal{C}_t : C \text{ splits } \pi(M)\}|] \leq 2^{3t/10}.$$

Proof. First note that M is split by exactly $\binom{t/2}{k/2}$ cuts of cardinality k since such a cut is formed by taking the union of exactly $k/2$ of the $t/2$ edges of M.

Denoting $\pi(C) = \{\pi(i) \in i \in C\}$ we have that C splits M if and only if $\pi(C)$ splits $\pi(M)$.

Thus, if C is a fixed cut of cardinality k, it splits $\pi(M)$ if and only if $\pi^{-1}(C)$ splits M. Since $\pi^{-1}(C)$ is uniformly over $\binom{[t]}{k}$, we see that

$$\Pr[C \text{ splits } \pi(M)] = \binom{t/2}{k/2}\binom{t}{k}^{-1} = \binom{t/2}{k/2}^{-1}.$$

Thus, the expected number $\mathbb{E}[|\{C \in C_t : C \text{ splits } \pi(M)\}|]$ of basis cuts that split $\pi(M)$ is at most

$$\leq \sum_{\substack{k=0 \\ k \text{ even}}}^{n} \left| C_t \cap \binom{[t]}{k} \right| \binom{t/2}{k/2} \Pr[C \text{ splits } \pi(M)]$$

$$\leq \sum_{\substack{k=0 \\ k \text{ even}}}^{n} \sum_{c=0}^{t/2} \binom{\frac{k}{2}}{c/2} \binom{\frac{t-k}{2}}{c/2} 2^c \binom{t/2}{k/2}^{-1}$$

$$\leq \sqrt{\max_{c \leq k} \binom{k}{c} \binom{t-k}{c} 4^c \binom{t}{k}^{-1}},$$

where we use Lemma 4.5 in the second inequality. If we substitute $\alpha := c/t, \beta := k/t$, take the base-two logarithm, divide by t, and use that

$$2^{h(\alpha)t - o(t)} \leq \binom{t}{\alpha t} \leq 2^{h(\alpha)t + o(t)},$$

we arrive at the following expression:

$$\tfrac{1}{2} \cdot \max_{0 \leq \alpha \leq \beta \leq 0.5} h\left(\frac{\alpha}{\beta}\right)\beta + h\left(\frac{\alpha}{1-\beta}\right)(1-\beta) + 2\alpha - h(\beta) < 3/10. \quad (2)$$

The inequality can be verified using a standard computer software such as Mathematica. See the full version for Mathematica sheet.[4] □

With all ingredients in place, Lemma 4.1 follows:

Proof of Lemma 4.1. We use Algorithm 2. It is easy to see that the $(\mathcal{A}_1, \mathcal{B}_1)$ forms an instance of HamPair that is equivalent with the instance $(\mathcal{A}, \mathcal{B})$. By Lemma 4.3 and a union bound on the two matchings that could form an Hamiltonian cycle, the instance $\mathcal{A}_2, \mathcal{B}_2$ is a YES-instance of HamPair with high probability if the instance $(\mathcal{A}_1, \mathcal{B}_1)$ is.

By Lemma 3.4, $res = u^T \mathbf{H}_t[\mathcal{A}_3, \mathcal{B}_3]v$. Thus, by Lemma 2.3 $res = 1$ with probability at least $1/4$ if $\mathbf{H}_t[\mathcal{A}_3, \mathcal{B}_3]$ is non-zero and there exists a solution. Moreover, if $\mathbf{H}_t[\mathcal{A}_3, \mathcal{B}_3]$ is the all-zero matrix then $res = 0$. This concludes the correctness.

For the runtime, note that Line 5 and Line 7 run in time

$$\sum_{A \in \mathcal{A} \cup \mathcal{B}} 2^{0.26t} + |\texttt{enumCuts(A)}|,$$

by Lemma 4.4. By Lemma 4.6 and the random permutation step, we have $\mathbb{E}[|\texttt{enumCuts(A)}|] \leq 2^{3t/10}$. Using Lemma 2.1 on Line 9 to make it run in $3^{t/2}t^{O(1)}$ time, the run time follows.

Note this only gives an expected run time guarantee, but by terminating the run time after n times its expectation we get a guaranteed run time probabilistic algorithm by Markov's inequality in a standard fashion. □

4.2 Weight Reduction

Proof of Theorem 2. Consider Algorithm 3. Note it works in general for weighted objects.[5] It is easy to see that the number of instances of HamPair generated is at most $O(\log(|\mathcal{A}| \cdot |\mathcal{B}|))$ since any pair $(A_i, a_i), (B_j, b_j)$ cannot be both in $\mathcal{A}_1 \times \mathcal{B}_2$ and in $\mathcal{B}_1 \times \mathcal{A}_2$ and the recursion depth is at most $O(\log(|\mathcal{A}| + |\mathcal{B}|))$. It follows that runtime of Algorithm 3 is at most $O((|\mathcal{A}| + |\mathcal{B}|)t^{O(1)})$.

Algorithm 3 Simple Weight Reduction Scheme.

Algorithm `list`$(\mathcal{A}, \mathcal{B}, t)$
Output: Instances $(\mathcal{A}_i, \mathcal{B}_i)_{i \leq \log n}$ with $\mathcal{A}_i \subseteq \mathcal{A}, \mathcal{B}_i \subseteq \mathcal{B}$ such that $(\mathcal{A}, \mathcal{B})$ is a YES-instance of HamPair if and only if $(\mathcal{A}_i, \mathcal{B}_i)$ is a YES-instance for some i.
1: Sort $\mathcal{A} = \{(A_1, a_1), \ldots, (A_l, a_l)\}$ such that $a_1 \leq a_2 \leq \ldots \leq a_l$
2: Let $\mathcal{A}_1 = \{(A_1, a_1), \ldots, (A_{\lfloor l/2 \rfloor}, a_{\lfloor l/2 \rfloor})\}$; $\mathcal{A}_2 = \mathcal{A} \setminus \mathcal{A}_1$
3: Sort $\mathcal{B} = \{(B_1, b_1), \ldots, (B_{l'}, b_{l'})\}$ suchothat $b_1 \leq b_2 \leq \ldots \leq b_{l'}$
4: Let k be the maximimum integer such that $a_l + b_k \leq t$
5: Let $\mathcal{B}_1 = \{(B_1, b_1), \ldots, (B_k, b_k)\}$; $\mathcal{B}_2 = \mathcal{B} \setminus \mathcal{B}_1$
6: **return** `list`$(\mathcal{A}_1, \mathcal{B}_2, t) \cup \{(\mathcal{A}_1, \mathcal{B}_1)\} \cup$ `list`$(\mathcal{A}_2, \mathcal{B}_1, t)$

For correctness, suppose there is a pair $P = ((A_i, w_i), (B_j, w_j))$ with $w_i + w_j \leq t$. If $i \leq \lfloor l/2 \rfloor$ and $j \leq k$ it is in $P \in \mathcal{A}_1 \times \mathcal{B}_1$. If $i \leq \lfloor l/2 \rfloor$ and $j > k$ then $P \in \mathcal{A}_1 \times \mathcal{B}_2$ and it will be covered in the recursive call with $\mathcal{A}_1$ and $\mathcal{B}_2$. Vice versa if $i > \lfloor l/2 \rfloor$ and $j \leq k$ then $P \in \mathcal{A}_2 \times \mathcal{B}_1$ and it will be covered in the recursive call with $\mathcal{A}_2$ and $\mathcal{B}_1$. Note P cannot be in $\mathcal{A}_2 \times \mathcal{B}_2$ since it would imply $w_i + w_j > t$.

Theorem 2 now follows directly from applying Algorithm 3 to reduce the given instance of MinHamPair to an instances of HamPair, and using Theorem 5 on the resulting instances. □

5 REDUCING (BIPARTITE) TSP TO MINHAMPAIR

In this section we use the algorithm from Theorem 2 as a blackbox to find fast algorithms for TSP. These reductions crucially rely on an algorithm proposed by Bodlaender et al. [BCKN15]. Intuitively their algorithm is a natural Dynamic Programming algorithm, but it uses a table reduction technique. It was shown in [BCKN15] that, if the recurrence associated with the dynamic programming algorithm only uses a fixed set of operations (defined below in Definition 5.2), then this technique can be automatically applied.

In this section we will need to slightly reorder the algorithm of [BCKN15], and therefore we will need several parts of their methods. While it will be very helpful for the reader to be familiar with [BCKN15], we made our presentation self-contained. and we recall the relevant parts from [BCKN15] in Subsection 5.1.

In Subsection 5.2 we present the proof of Theorem 1, and in Subsection 5.3 we prove Theorem 3.

5.1 Tools and Notations From [BCKN15]

Given a base set U, we use $\Pi(U)$ for the set of all partitions of U. It is known that, together with the coarsening relation $\sqsubseteq$, $\Pi(U)$ gives a lattice, with the minimum element being $\{U\}$ and the maximum element being the partition into singletons. We denote $\sqcap$ for the meet operation and $\sqcup$ for the join operation in this lattice; these operators are associative and commutative. If $X \subseteq U$ we let $p_{\downarrow X} \in \Pi(X)$ be the partition obtained by removing all elements not in X from it, and analogously we let for $U \subseteq X$ denote $p_{\uparrow X} \in \Pi(X)$ for the partition obtained by adding singletons for every element in

[4] Unfortunately, we were not able to show this analytically.
[5] We are not aware of previous work where this natural algorithm appeared, but similar ideas were used in other weight-reduction schemes [WW18, NvLvdZ12].

$X \setminus U$ to p. Also, for $X \subseteq U$, we let $U[X]$ be the partition of U where one block is $\{X\}$ and all other blocks are singletons. If $a, b \in U$ we shorthand $U[ab] = U[\{a, b\}]$. The empty set, vector and partition are all denoted by $\emptyset$.

Given a set $X \subseteq E$, we denote $N_X(v)$ for the neighbors of v in the graph with edge set X, and we denote $\deg_X(v) = |N_X(v)|$.

Definition 5.1 (set of weighted partitions). *A set of weighted partitions is a set $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$, i.e., a family of pairs, each consisting of a partition of U and a non-negative integer weight.*

For notational ease, we let $\texttt{rmc}(\mathcal{A})$ denote the set obtained by removing non-minimal weight copies:

$$\texttt{rmc}(\mathcal{A}) = \left\{(p, w) \in \mathcal{A} \,\middle|\, \nexists (p, w') \in \mathcal{A} \wedge w' < w\right\}.$$

Definition 5.2 (operators on weighted partitions). *Let U be a set and $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$.*

Union. *For $\mathcal{B} \subseteq \Pi(U) \times \mathbb{N}$, define $\mathcal{A} \Cup \mathcal{B} = \texttt{rmc}(\mathcal{A} \cup \mathcal{B})$.* Combine two sets of weighted partitions and discard dominated partitions.

Insert. *For $X \cap U = \emptyset$, let $\texttt{ins}(X, \mathcal{A}) = \left\{(p_{\uparrow U \cup X}, w) \,\middle|\, (p, w) \in \mathcal{A}\right\}$.* Insert additional elements into U and add them as singletons in each partition.

Shift. *For $w' \in \mathbb{N}$ define $\texttt{shft}(w', \mathcal{A}) = \{(p, w + w') \mid (p, w) \in \mathcal{A}\}$.* Increase partition's weight by w'.

Glue. *For u, v, let $\hat{U} = U \cup \{u, v\}$ and let $\texttt{glue}(uv, \mathcal{A}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$*

$$\texttt{glue}(uv, \mathcal{A}) = \texttt{rmc}(\left\{(\hat{U}[uv] \sqcap p_{\uparrow \hat{U}}, w) \,\middle|\, (p, w) \in \mathcal{A}\right\}).$$

Moreover, if $w : \hat{U} \times \hat{U} \to \mathbb{N}$, then we define $\texttt{glue}_w(uv, \mathcal{A}) = \texttt{shft}(w(u, v), \texttt{glue}(uv, \mathcal{A}))$. In each partition combine the sets containing u and v into one; add u and v to the base set if needed.

Project. *For $X \subseteq U$ let $\overline{X} = U \setminus X$, and define $\texttt{proj}(X, \mathcal{A}) \subseteq \Pi(\overline{X}) \times \mathbb{N}$ as $\texttt{proj}(X, \mathcal{A}) =$*

$$\texttt{rmc}(\left\{(p_{\downarrow \overline{X}}, w) \,\middle|\, (p, w) \in \mathcal{A} \wedge \forall e \in X : \exists e' \in \overline{X} : p \sqsubseteq U[ee']\right\}).$$

Remove all elements of X from each partition, but discard partitions where this would reduce the number of blocks/sets.

Join. *For $\mathcal{B} \subseteq \Pi(U') \times \mathbb{N}$ let $\hat{U} = U \cup U'$ and define $\texttt{join}(\mathcal{A}, \mathcal{B}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as $\texttt{join}(\mathcal{A}, \mathcal{B}) =$*

$$\texttt{rmc}(\left\{(p_{\uparrow \hat{U}} \sqcap q_{\uparrow \hat{U}}, w_1 + w_2) \,\middle|\, (p, w_1) \in \mathcal{A} \wedge (q, w_2) \in \mathcal{B}\right\}).$$

Extend all partitions to the same base set. For each pair of partitions return the outcome of the meet operation $\sqcap$, with weight equal to the sum of the weights.

Definition 5.3 (Preserving representation). *A function $f : 2^{\Pi(U) \times \mathbb{N}} \times Z \to 2^{\Pi(U') \times \mathbb{N}}$ is said to* preserve representation *if for every $\mathcal{A}, \mathcal{A}' \subseteq \Pi(U) \times \mathbb{N}$ and $z \in Z$ it holds that if $\mathcal{A}'$ represents $\mathcal{A}$ then $f(\mathcal{A}', z)$ represents $f(\mathcal{A}, z)$. (Note that Z stands for any combination of further inputs.)*

Lemma 5.4 ([BCKN15]). *Each of the operations union, shift, glue and project can be performed in $S|U|^{O(1)}$ time where S is the size of the input of the operation. Given $\mathcal{A}, \mathcal{B}$, $\texttt{join}(\mathcal{A}, \mathcal{B})$ can be computed in time $|\mathcal{A}| \cdot |\mathcal{B}| \cdot |U|^{O(1)}$.*

5.2 Splitting the Bipartite Graph into Subgraphs

In this section we prove Theorem 1. As mentioned in the introduction, the basic idea of the reduction is as follows: The starting point is an algorithm from [BCKN15] that solves TSP in $O((2+2^{\omega/2})^{\mathrm{pw}} n)$ time. Note that this algorithm already solves TSP on bipartite graphs in $2^n n^{O(1)}$ if $\omega = 2$. Moreover, the algorithm only computes $O((2+\sqrt{2})^{n/2} n^{O(1)})$ bits of data, and the only bottleneck of the algorithm is the computation of representative sets.

Let $G = (L \cup R, E)$ be an undirected bipartite graph and let $w : E \to \mathbb{R}$. We can assume that $|L| = |R| = n/2$, as otherwise G has no Hamiltonian cycle. A bipartite graph has a special path decomposition in which each bag contains L and the vertices in R are one by one introduced and forgotten. Our approach is to compute all table entries corresponding to the the sub-problem where half (say R') of the vertices are introduced. Afterwards we do the same for $R \setminus R'$. To compute all these table entries within the time bound we slightly lower the frequency of calls to the procedure `reducematchings` from Theorem 4. Concretely, we split R' in R_1 and R_2, where $|R_1| = \alpha n/2$ and $|R_2| = (\frac{1}{2} - \alpha)n/2$ and α will be a small constant. Then we find representative sets for sub-solutions using R_1 and R_2 and take all combinations *without* reducing the number of sub-solutions with `reducematchings` afterwards.

While this avoid the computational bottleneck in the run time, it causes different problems as an increase in sub-solutions also increase the run time. Here Theorem 2 comes to the rescue: Given the sub-solutions using both R and R', it finds the minimum weight pair fast enough to compensate for the increased number of sub-solutions.

To guide the dynamic programming algorithm we will use the following definition:

Table 1: Probabilities of each type.

$(1,2)$-type τ	$(2,0)$	$(0,2)$	$(1,1)$	$(1,0)$	$(0,1)$	$(0,0)$
Probability p_τ	α^2	$(\frac{1}{2}-\alpha)^2$	$2\alpha(\frac{1}{2}-\alpha)$	α	$2(\frac{1}{2}-\alpha)\frac{1}{2}$	$\frac{1}{4}$
$(3,4)$-type τ	$(2,0)$	$(0,2)$	$(1,1)$	$(1,0)$	$(0,1)$	$(0,0)$
Probability p_τ	α^2	$(\frac{1}{2}-\alpha)^2$	$2\alpha(\frac{1}{2}-\alpha)$	α	$2(\frac{1}{2}-\alpha)\frac{1}{2}$	$\frac{1}{4}$

Definition 5.5. *An α-partition is a partition ρ of R into $R_1 \cup R_2 \cup R_3 \cup R_4$ such that*

$$|R_1|, |R_3| \in \alpha n \pm 10\sqrt{n} \quad \text{and} \quad |R_2|, |R_4| \in \alpha n \pm 10\sqrt{n}. \tag{3}$$

Let $C \subseteq E$. Fix a partition $\rho = (R_1, R_2, R_3, R_4)$ of R. Define the $(1,2)$-type of a vertex l as $(|N_C(l) \cap R_1|, |N_C(l) \cap R_2|)$. Similarly, define the $(3,4)$-type of a vertex l as $(|N_C(l) \cap R_3|, |N_C(l) \cap R_4|)$. Say ρ is α-regular for C if it is α partition and the following condition hold for all τ and p_τ as listed in Table 1:

$$\begin{aligned} |\{l \in L : l \text{ has } (1,2)\text{-type } \tau\}| &\in (p_\tau n/2 \pm 10\sqrt{n}), \\ |\{l \in L : l \text{ has } (3,4)\text{-type } \tau\}| &\in (p_\tau n/2 \pm 10\sqrt{n}). \end{aligned} \tag{4}$$

By upper bounding the probability that (3) and (4) fail with standard Chernoff bounds[6] and using the union bound, the following observation can be obtained:

[6] If $a_1, \ldots, a_n$ are independent and Bernoulli and $X = a_1 + a_2 + \ldots + a_n$, then $\Pr[|X - E[x]| \geq t] \leq 2\exp(-2t^2/n)$.

Observation 5.6. *Let C be a Hamiltonian cycle of G. And let $\rho = (R_1, R_2, R_3, R_4)$ be obtained by adding each vertex of R independently to*

$$R_1 \text{ with probability } \alpha, \qquad R_2 \text{ with probability } \tfrac{1}{2} - \alpha,$$
$$R_3 \text{ with probability } \alpha, \qquad R_4 \text{ with probability } \tfrac{1}{2} - \alpha.$$

Then ρ is α-regular for C with at least constant probability.

For a subset $Z \subseteq R$ we let $G_Z = (V_Z, E_Z)$ be the graph $G[L \cup Z]$, and we define $A_Z(s) =$

$$\{(M, \min_{X \in \mathcal{E}(M,s)} w(X)) : M \in \Pi_{\mathrm{m}}(s^{-1}(1)) \wedge \mathcal{E}_Z(M,s) = \emptyset\},$$
$$\mathcal{E}_Z(M,s) = \big\{X \subseteq E_Z : v \in L \to \deg_X(v) = s(v),$$
$$\text{and } \forall j \in Z : v \in \deg_X(r'_j) = 2,$$
$$\text{and } \{u,v\} \in M \to u \text{ and } v \text{ are connected in } G[X],$$
$$\text{and } G[X] \text{ contains no cycles}\big\}.$$

Note this definition is the same as the corresponding definition in Section [BCKN15, Section 3.4] in the special case that the path decomposition has L in each bag and has forgotten set Z from R and not yet introduced $R \setminus Z$. It will also be useful to refer to a degree-sequence as regular if it acts as expected:

Definition 5.7. *We say a vector $s \in \{0,1,2\}^L$ is α-regular if*

$$|s^{-1}(0)| \in (1-\alpha)^2 n/2 \pm 10\sqrt{n}, \quad |s^{-1}(1)| \in \alpha(1-\alpha)n/2 \pm 10\sqrt{n}$$
$$|s^{-1}(2)| \in \alpha^2 n/2 \pm 10\sqrt{n}.$$

Lemma 5.8. *Let $|Z| = \alpha n/2$ where $\alpha < \frac{1}{2}$. There is an algorithm that computes for every α-regular $s \in \{0,1,2\}^L$ a family of weighted matchings $A'_Z(s)$ that represents $A_Z(s)$ such that $|A_Z(s)| \leq 2^{|s^{-1}(1)|/2}$. The algorithm runs in time*

$$\sum_{\alpha\text{-regular } s} \binom{n/2}{|s^{-1}(0)|, |s^{-1}(1)|, |s^{-1}(2)|} 2^{\omega |s^{-1}(1)|/2}.$$

The proof of Lemma 5.8 is a straightforward adaption of the algorithm described in [BCKN15, Section 3.4]. A self-contained proof is given in Appendix **??**

Theorem 1 (restated). *Suppose that $(s \times s)$-matrices can be multiplied in $s^{2+o(1)}$ time. Then there is a Monte Carlo algorithm that, given an undirected bipartite graph $G = (L \cup R, E)$ and weights $w : E \to \mathbb{R}$, finds a Hamiltonian tour of minimum weight in $O(1.9999^n)$ time.*

Proof. Let $\alpha = 0.493526$, and note that by our assumption we may use $\omega = 2$. Let C be a Hamiltonian cycle of minimum weight. Pick a random partition $\rho = (R_1, R_2, R_3, R_4)$ as in Observation 5.6. This splits the Hamiltonian cycle in $C_i = C \cap E_{R_i}$

Thus ρ is α-regular for C with at least constant probability.

This means that if $s(v) = d_{C_i}(v)$ for every $v \in L$, then $A_{R_i}(s)$ contains the matching obtained by contracting all vertices in $s^{-1}(2)$ in C_i, and it will have associated weight $w(C \cap E_{R_i})$. Moreover, s_1, s_3 will be α-regular and s_2, s_4 will be $(\frac{1}{2} - \alpha)$ regular since ρ is α-regular for C.

We use Lemma 5.8 to compute tables A'_{R_i} representing A_{R_i} for $i = 1, 2, 3, 4$. The runtime will be

$$\binom{n/2}{\alpha^2, 2\alpha(1-\alpha), (1-\alpha)^2} 2^{2\alpha(1-\alpha)n/2} \leq 1.9999^n,$$

where the upper bound follows from a simple Mathematica computation. Similarly to the case of the join bag in [BCKN15, Section 3.4], it holds that $A_{R_1 \cup R_2}(s)$

$$\bigcup_{l+r=s} \mathtt{proj}(s^{-1}(2) \setminus (l^{-1}(2) \cup r^{-1}(2)), \mathtt{join}(A_{R_1}(l), A_{R_2}(r))). \quad (5)$$

Note here that $l, r, s \in \{0,1,2\}^L$ hence the summation is vector summation. Since we combine two characteristics of partial solutions into a new one, the degrees of the left and right partial solution (l and r) have to sum up to the degrees of the new one (s). Two characteristics $(M_1, w_1) \in A_{R_1}(l)$ and $(M_2, w_2) \in A_{R_2}(r)$ combine to a characteristic of $A_{R_1 \cup R_2}(s)$ if and only if $M_1 \cup M_2$ is acyclic which is equivalent to saying that all vertices in $s^{-1}(2)$ are connected to vertices in $s^{-1}(1)$ in the resulting partition $M_1 \sqcap M_2$, and the latter is ensured by the project operation.

Note that (5) uses only operations that preserve representation by Lemma 5.4. Using the tables A'_{R_i} representing A_{R_i} for $i = 1, 2, 3, 4$ we can compute tables $A'_{R_1 \cup R_2}$ and $A'_{R_3 \cup R_4}$ that represent $A_{R_1 \cup R_2}$ and $A_{R_3 \cup R_4}$ respectively.

Moreover, we can even restrict the evaluation according to (5) by only iterating over α-regular l and $(\frac{1}{2} - \alpha)$-regular r by the assumption that ρ is α-regular for C. This means that $|l^{-1}(1)| = 2\alpha(1-\alpha)$ and $|r^{-1}(1)| = 2(\frac{1}{2} - \alpha)(\frac{1}{2} + \alpha)$.

In summary, we can compute $A'_{R_1 \cup R_2}(s) =$

$$\bigcup_{\substack{l+r=s \\ l \text{ is } \alpha\text{-regular} \\ r \text{ is } (1/2-\alpha) \text{ regular}}} \mathtt{proj}(s^{-1}(2) \setminus (l^{-1}(2) \cup r^{-1}(2)), \mathtt{join}(A_{R_1}(l), A_{R_2}(r))).$$

Using this expression for $A'_{R_1 \cup R_2}(s)$ and observing $\mathtt{join}(\mathcal{A}, \mathcal{B})$ can be computed in time $|\mathcal{A}| \cdot |\mathcal{B}| \cdot |U|^{O(1)}$ (by Lemma 5.4), we obtain the following bound: $\sum_{s \text{ is } \alpha\text{-regular}} |A'_{R_1 \cup R_2}(s)|$ is at most

$$\leq \binom{n/2}{\alpha^2 n/2, (\frac{1}{2} - \alpha)^2 n/2, 2a(\frac{1}{2} - \alpha)n/2, \alpha n/2, 2(\frac{1}{2} - \alpha)\frac{1}{2}n/2, \frac{1}{4}n/2} \cdot 2^{(|l^{-1}(1)| + |r^{-1}(1)|)n/4}.$$

Here the multinomial coefficient counts the number of ways of combinations of l and r using Table 1 and $2^{|l^{-1}(1)|n/4}$ and $2^{|r^{-1}(1)|n/4}$ count the number of matchings in the table $A_{R_i}(l)$ and $A_{R_i}(r)$. By the definition of α-regularity $|l^{-1}(1)| = 2\alpha(1-\alpha)$ and $|r^{-1}(1)| = 2(1/2 - \alpha)(1/2 + \alpha)$. Evaluating the upper bound for the chosen value of α shows

$$\sum_{s \text{ is } \alpha\text{-regular}} |A'_{R_1 \cup R_2}(s)| \leq 2^{0.924879n}.$$

Similarly, we can also compute $A_{R_1 \cup R_2}(s)$ that represents $A_{R_3 \cup R_4}(s)$.

Now it remains to iterate over all s and search for the minimum weight combination of perfect matchings in $A'_{R_1 \cup R_2}(s)$ and $A'_{R_3 \cup R_4}(\vec{2} - s)$, where $\vec{2}$ denotes the vector with all values equal to

2. Using Theorem 2 we see that the total runtime becomes

$$\Big(\sum_{s \text{ is } \alpha-\text{regular}} (|A'_{R_1\cup R_2}| + |A'_{R_3\cup R_4}|)2^{3|s^{-1}(1)|/10}\Big) + \Big(\sum_{s \text{ is } \alpha-\text{regular}} 3^{|s^{-1}(1)|/2}\Big) \leq 2^{0.9999n},$$

where we use $|s^{-1}(1)| = n/4$ since s is α-regular. □

5.3 Towards Faster TSP in Non-Bipartite Graphs

In this subsection we prove Theorem 3. Suppose the (not necessarily bipartite) undirected graph $G = (V, E)$ and weight function $w : E \to \mathbb{R}$ form an instance of the TSP problem. We can assume without loss of generality that $n := |V|$ is an even number by an easy reduction. Then the optimal solution tour T can be decomposed in two perfect matchings M_1 and M_2.

Let $M_0 \in_R \Pi(V)$. For an edge set $X \subseteq E$ we define $V(X) = \cup_{\{u,v\}\in X}\{u, v\}$. We focus on computing a set of matchings that represents all matchings that form an Hamiltonian cycle with M_0. To this end we arbitrarily fix $s \in V$, and define for $X \subseteq M_0$, and t such that $s \notin V(X)$ and $t \in V(X)$:

$$A_t(X) = \{(M, w(M)) : M \in \Pi_m(V(X) \cup \{s\} \setminus \{t\}), \\ M \cup X \text{ acyclic and connects } s \text{ to } t\}$$

If $|X| \leq 1$, it is easily seen that $A_t(X) = \{(\emptyset, 0)\}$ if $s = t$ and X is only one edge that contains t, and $A_t(X) = \emptyset$ otherwise. For $|X| > 1$ we have the following recurrence:

$$A_t(X) = \bigcup_{t'\in V(X)\setminus\{t,\alpha_{M_0}(t)\}}^{\downarrow} \texttt{glue}_w(\{t, \alpha_{M_0}(t')\}, \\ \texttt{ins}(\{t, \alpha_{M_0}(t')\}, A_{t'}(X \setminus \{t, \alpha_{M_0}(t)\}))). \quad (6)$$

To see that this recurrence holds, recall that for any matching $M \in A_t(X)$ the edge set $M \cup X$ forms a path from s to t, and it must use edges from M and M_0 in an alternating fashion. Therefore penultimate vertex of this path is $\alpha_{M_0}(t)$. The recurrence tries all possibilities of the predecessor t' of $\alpha_{M_0}(t)$. If t' is such a predecessor, the matching M is formed from a matching $M' \in A_{t'}(X \setminus \{t, \alpha_{M_0}(t)\})$ and the edge $\{t', \alpha_{M_0}(t)\}$. Thus the right-hand side of (6) indeed computes all matchings in $A_t(X)$.

Note that $A_s(M_0)$ contains all matchings that form an Hamiltonian cycle.

Let α, β be such that $\alpha < 0.7$ and $\alpha + \beta < \log(3)/2$. Assume there is an algorithm `reducematching` that, given an n-vertex undirected graph G with edge weights $w : E(G) \to \mathbb{R}$ and a family of perfect matchings $\mathcal{A} \subseteq \Pi_m(V(G))$, computes a set $\mathcal{A}' \subseteq \mathcal{A}$ that represents $\mathcal{A}$ and satisfies $|\mathcal{A}'| \leq 2^{\alpha n}$ in $|\mathcal{A}|2^{\beta n}$ time.

We can build a representative set of $A_s(M_0)$ using (6) an applying the assumed `reducematchings` procedure since all operations used preserve representation by Lemma 5.4. The runtime of this operation would be

$$\sum_{t\in V(X)}\sum_{|X|=0}^{n/2}\binom{n/2}{|X|}|A_t(X)|2^{2\beta|X|} \leq \sum_{|X|=0}^{n/2}\binom{n/2}{|X|}2^{2(\alpha+\beta)|X|}n^{O(1)} \\ \leq (1 + 2^{\alpha+\beta})^{n/2}n^{O(1)} \leq (1 + 3^{1-\varepsilon})^{n/2}n^{O(1)},$$

and this is $2^{(1-\varepsilon')n}$ for some $\varepsilon' > 0$ that depends on ε.

It is well known that the probability that two random perfect matchings on the complete graph on n vertices form an Hamiltonian cycle is at least $1/n^{O(1)}$.

Thus, if the optimal tour is $M_1 \cup M_2$, and we sample $n^{O(1)}$ random perfect matchings M_0 and take $\mathcal{A}$ to be union of all obtained representing sets, then this set will represent both M_1 and M_2 with high probability. Thus it remains to solve the instance $(\mathcal{A}, \mathcal{A})$ of MinHamPair.

The size of $\mathcal{A}$ will be $n^{O(1)}2^{\alpha n}$, and since by assumption $\alpha < 0.7 - \varepsilon$, Theorem 3 follows from Theorem 2.

ACKNOWLEDGEMENTS.

The author would like to thank Nikhil Bansal for some inspiring discussions.

A MERLIN-ARTHUR PROTOCOLS

We briefly elaborate[7] the claim that there exist $O(1.9999^n)$ verification time Merlin-Arthur protocols for bipartite TSP. Specifically, we claim a prover can design a proof on $O(1.9999^n)$ bits that a given bipartite TSP instance has no tour of weight at most t, and this proof can be verified by a probabilistic verfied using $O(1.9999^n)$ time. The verifier will always agree with the prover is the proof was correct and there is no tour of weight at most t. On the other hand, if there is tour of weight at most t the prover will not accept the proof of the non-existence of such a tour with at least constant probability. We refer to [Wil16, BK16] for further definition of Merlin-Arthur proof systems.

To turn the algorithm behind Theorem 1 into a Merlin-Arthur proof system we use the following adaptions:

- Observation 5.6 is derandomized by letting $\mathcal{F} = \{f \to \{1, 2, 3, 4\}\}$ be an appropriate pair-wise independent hash function and letting $R_i = f^{i-1}(i)$. Observation 5.6 can still be proven to hold by bounding the variance and applying Chebyshev's inequality.
- For each $f \in \mathcal{F}$, Merlin computes the appropriate representative sets and sends them to Arthur. He also provides for each row basis computation underlying each invocation of `reducematchings` the according PLUQ decomposition (see [DLP17, Theorem 1]).
- Arthur verifies all row basis computations using Freivalds algorithm (see [DLP17, Theorem 1]).
- Arthur runs the randomized algorithm from Theorem 2 to verify that indeed there is no pair of matchings of weight at most t.

REFERENCES

[ABI+19] Andris Ambainis, Kaspars Balodis, Janis Iraids, Martins Kokainis, Krisjanis Prusis, and Jevgenijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1783–1793, 2019.

[Alm19] Josh Alman. Limits on the universal method for matrix multiplication. In *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA.*, pages 12:1–12:24, 2019.

[7] We believe this result is of minor importance but nevertheless sketch how to prove it since it is independent of whether $\omega = 2$.

[BCKN15] 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.

[Bel62] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962.

[BHKK09] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting paths and packings in halves. In *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, pages 578–586, 2009.

[BHKK12] 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.

[BHKK17] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017.

[Bjö14] Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014.

[BK16] Andreas Björklund and Petteri Kaski. How proofs are prepared at camelot: Extended abstract. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 391–400, 2016.

[BKK17] Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed hamiltonicity and out-branchings via generalized laplacians. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 91:1–91:14, 2017.

[Cha13] Timothy M. Chan. The art of shaving logs. In *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, page 231, 2013.

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[CKN18] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018.

[CKSU05] Henry Cohn, Robert D. Kleinberg, Balázs Szegedy, and Christopher Umans. Group-theoretic algorithms for matrix multiplication. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 379–388, 2005.

[CLN18] Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting hamiltonian cycles via matrix rank. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099, 2018.

[CNP+11] 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 *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159, 2011.

[Coo11] William J Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, Princeton, NJ, 2011.

[CP15] Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. *Inf. Comput.*, 243:75–85, 2015.

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[DLP17] Jean-Guillaume Dumas, David Lucas, and Clément Pernet. Certificates for triangular equivalence and rank profiles. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 133–140, New York, NY, USA, 2017. ACM.

[Epp07] David Eppstein. The traveling salesman problem for cubic graphs. *J. Graph Algorithms Appl.*, 11(1):61–81, 2007.

[FLPS16] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016.

[Fre77] Rusins Freivalds. Probabilistic machines can use less running time. In *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977.*, pages 839–842, 1977.

[Gal14] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

[Geb08] Heidi Gebauer. On the number of hamilton cycles in bounded degree graphs. In *Proceedings of the Fifth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2008, San Francisco, California, USA, January 19, 2008*, pages 241–248, 2008.

[GS87] Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987.

[HK62] Michael. Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[IN07] Kazuo Iwama and Takuya Nakashima. An improved exact algorithm for cubic graph TSP. In *Computing and Combinatorics, 13th Annual International Conference, COCOON 2007, Banff, Canada, July 16-19, 2007, Proceedings*, pages 108–117, 2007.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[Kar82] Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1(2):49–51, 1982.

[Kas18] Petteri Kaski. Engineering a delegatable and error-tolerant algorithm for counting small subgraphs. In *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018.*, pages 184–198, 2018.

[KGK77] 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, Seattle, Washington, USA, October 16-19, 1977*, pages 294–300, 1977.

[Kou08] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 575–586, 2008.

[KW16a] Ioannis Koutis and Ryan Williams. Algebraic fingerprints for faster algorithms. *Commun. ACM*, 59(1):98–105, 2016.

[KW16b] Ioannis Koutis and Ryan Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, 2016.

[LN10] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 321–330, 2010.

[NvLvdZ12] Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. Reducing a target interval to a few exact queries. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, pages 718–727, 2012.

[RS95] Ran Raz and Boris Spieker. On the "log rank"-conjecture in communication complexity. *Combinatorica*, 15(4):567–588, 1995.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[Sch05] Alexander Schrijver. On the history of combinatorial optimization (till 1960). In K. Aardal, G.L. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 1 – 68. Elsevier, 2005.

[Ser78] Anatoliy I. Serdyukov. On some extremal tours in graphs (in russian). *Upravlyaemye systemy*, 17:76–79, 1978.

[Str94] Volker Strassen. Algebra and complexity. In *First European Congress of Mathematics Paris, July 6–10, 1992*, pages 429–446. Springer, 1994.

[STV18] Ola Svensson, Jakub Tarnawski, and László A. Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 204–213, 2018.

[Tut47] William T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, s1-22(2):107–111, 1947.

[TV19] Vera Traub and Jens Vygen. Approaching 3/2 for the s-t-path TSP. *J. ACM*, 66(2):14:1–14:17, 2019.

[Wil09] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

[Wil16] Ryan Williams. Strong ETH breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 2:1–2:17, 2016.

[Wil18] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018.

[Wlo19] Michal Wlodarczyk. Clifford algebras meet tree decompositions. *Algorithmica*, 81(2):497–518, 2019.

[WW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

[Yat37] Franck Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science. Technical Communication. Imperial Bureau of Soil Science, 1937.

[Zen19] Rico Zenklusen. A 1.5-approximation for path TSP. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1539–1549, 2019.