Exploiting Sparsity for Bipartite Hamiltonicity

Andreas Björklund
Department of Computer Science, Lund University, Sweden

Abstract

We present a Monte Carlo algorithm that detects the presence of a Hamiltonian cycle in an n-vertex undirected bipartite graph of average degree $\delta \geq 3$ almost surely and with no false positives, in $(2-2^{1-\delta})^{n/2}\,\mathrm{poly}(n)$ time using only polynomial space. With the exception of cubic graphs, this is faster than the best previously known algorithms. Our method is a combination of a variant of Björklund's $2^{n/2}\,\mathrm{poly}(n)$ time Monte Carlo algorithm for Hamiltonicity detection in bipartite graphs, SICOMP 2014, and a simple fast solution listing algorithm for very sparse CNF-SAT formulas.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Hamiltonian cycle, bipartite graph

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.3

Funding This work was supported in part by the Swedish Research Council grant VR-2016-03855, "Algebraic Graph Algorithms".

1 Introduction

Given an n-vertex undirected graph $G = (V, E)$, a Hamiltonian cycle is a vertex permutation $(v_1, v_2, \ldots, v_n)$ such that $v_i v_{i+1} \in E$ for all i, including also $v_n v_1 \in E$. The algorithmic problem of deciding if a graph has a Hamiltonian cycle was one of the first problems recognised as NP-hard [12]. For general graphs, an $O(1.657^n)$ time algorithm is known [1]. A natural question to ask is if one can do better in sparse graphs, since the average number of entry and exit alternatives for the cycle at a vertex is smaller. Cygan et al. [5] proved a $(2+\sqrt{2})^{\mathrm{pw}(P_G)}\,\mathrm{poly}(n)$ time algorithm that detects a Hamiltonian cycle given a path-decomposition P_G of the graph G of width $\mathrm{pw}(P_G)$. In sparse graphs of average degree δ, path-decompositions of width at most $\delta n/11.538$ can be found in polynomial time as proved by Kneis et al. [13]. Hence the combination of these two results gives faster algorithms for Hamiltonicity in sparse undirected graphs when $\delta < 4.73$. However, the techniques in both [5] and [13] do not seem to directly give much faster algorithms when we guarantee that the graph in addition of being sparse is also bipartite. In contrast, there is a much faster $2^{n/2}\,\mathrm{poly}(n) \subset O(1.415^n)$ time algorithm for general bipartite graphs [1]. In this paper we propose a method to speed up the latter algorithm to get faster algorithms in sparse bipartite graphs. Our main result says

▶ **Theorem 1.** *There is a Monte Carlo algorithm that given an undirected bipartite graph on n vertices and average degree $\delta \geq 3$ detects if it has a Hamiltonian cycle in* $\mathrm{poly}(n)$ *space and*

$$(2-2^{1-\delta})^{n/2}\,\mathrm{poly}(n)$$

time, without false positives and false negatives with probability at most 2^{-n}.

See Table 1 for some typical running time bases.

Table 1 The base c in the running time bound c^n of our algorithm for the first small δ's.

δ	3	3.25	3.5	3.75	4	4.25	4.5	4.75	5	5.25
c	1.3229	1.3378	1.3503	1.3606	1.3693	1.3765	1.3826	1.3877	1.3919	1.3955

As far as we know, this is the first example of a Hamiltonicity algorithm that operates in less than $2^{n/2}$ time for sparse bipartite graphs, save for the special case of cubic (3-regular) graphs for which much faster algorithms have been found. However, the techniques used in these cubic graph algorithms do not scale gracefully with the average degree and already for 4-regular graphs our algorithm is much faster than previous ones. Confer the related work section below for more discussion of earlier algorithms. We also note that our algorithm can be modified to compute the parity of the number of Hamiltonian cycles:

▶ **Theorem 2.** *There is a Las Vegas algorithm that given an undirected bipartite graph on n vertices and average degree $\delta \geq 3$ computes the parity of the number of Hamiltonian cycles in* $\operatorname{poly}(n)$ *space and*

$$(2 - 2^{1-\delta})^{n/2} \operatorname{poly}(n)$$

expected time.

The combination of techniques employed by our algorithms is similar to the overall idea in the algorithm by Björklund and Husfeldt [2] to compute the parity of the number of Hamiltonian cycles, and subsequently the modular counting algorithm for Hamiltonian cycles in Björklund et al. [4]. The idea is as follows: We first devise an algebraic fingerprint for Hamiltonian cycles, i.e., an exponential sum S with variables on the edges of the graph such that S evaluates to a non-zero value only if the underlying graph has a Hamiltonian cycle. Furthermore, if it has a Hamiltonian cycle, it evaluates to a non-zero value with large probability under a random assignment to the variables. Next we show that the exponential sum S can in fact be randomly chosen from a family of exponential sums, all of which evaluate to the same value. In a randomly chosen exponential sum in the family only a small fraction of the exponentially many terms contribute non-zero values in expectation. If we only knew which those terms were, we could evaluate the exponential sum much faster than summing over all terms. To this end, we show that listing a small superset of the terms that evaluate to non-zero values can be done by means of another exponential time algorithm. In our case here we can encode the interesting terms as solutions to a very sparse CNF-SAT formula. These solutions can in turn be listed by a branching algorithm in combination with a perfect matching algorithm. An alternative listing algorithm for the most interesting values $\delta \leq 5.5$ can also be done by a dynamic programming algorithm across a path decomposition of the variable/clause incidence graph of the formula.

1.1 Related Work

Several diverse ideas for Hamiltonicity detection in general and sparse graphs have been pursued to get improved worst case running time bounds. We give here a brief list of the results we are aware of.

1.1.1 Dynamic programming across a path decomposition

Cygan et al. [5] prove that one can decide the existence of Hamiltonian cycles in undirected graphs in $(2 + \sqrt{2})^{\operatorname{pw}(P_G)} \operatorname{poly}(n)$ time, where $\operatorname{pw}(P_G)$ is the path-width of any path-decomposition P_G of G given as input. It relies on dynamic programming across the path decomposition and hence requires exponential space usage in $\operatorname{pw}(P_G)$.

Using the best known bound on the path-width for graphs of average degree δ by Kneis et al. [13], which says that one can construct a path-decomposition P with $\mathrm{pw}(P_G) \leq \frac{\delta n}{11.538}$, we get an $O(1.0619^{\delta n})$ time bound. This running time is worse than ours for all $\delta \geq 3$. However, in graphs of maximum degree three, another algorithm of Cygan et al. [6] can be accelerated to run in $3^{\mathrm{pw}(G)}\,\mathrm{poly}(n)$ time. It uses the efficient path decomposition in cubic graphs devised by Fomin and Høie [10], to arrive at an $O(1.201^n)$ time algorithm. We note that the efficient path-decomposition for cubic graphs of Fomin and Høie [10] is in fact used by the path-decomposition of Kneis et al. [13] in combination with branching. We also note that the very fast algorithm for cubic graphs above is the result not only of the efficient path-decomposition bound in cubic graphs, but also the fact that one only needs 3 states per vertex in a bag for cubic graphs as opposed to the (amortised) $2 + \sqrt{2}$ number of states per vertex in the general algorithm of Cygan et al. [5] (Confer Corollary 1.6 in Cygan et al. [6]). Hence, the cubic case is indeed special for this approach.

1.1.2 Branching

A very natural approach to Hamiltonicity detection in cubic graphs is to guess which two of the three edges incident to a vertex are used on the cycle and branch. This will in turn diminish the number of alternatives for how the cycle can pass through the three neighbor vertices. There is an $O(1.251^n)$ algorithm for Hamiltonicity decision in graphs of maximum degree three based on carefully analysed branching [11]. It improves slightly over the $O(1.260^n)$ time algorithm by Eppstein [9]. The algorithms in fact work for the Travelling Salesman problem solving for the minimum cost edge-weighted Hamiltonian cycle. Eppstein also provides an $O(1.297^n)$ time algorithm that can list the solutions and hence determine their number. He also exhibits bipartite maximum degree three graphs that has $\Omega(1.260^n)$ Hamiltonian cycles, demonstrating that any algorithm that enumerates the cycles one-by-one must take this long in the worst case.

1.1.3 Directed algorithms

For directed bipartite graphs, there is an $O(1.733^n)$ time algorithm [4]. However, it is still open whether there exists an $O(c^n)$ time algorithm for some $c < 2$ for decision in directed graphs. In directed graphs of average degree δ, counting the Hamiltonian cycles can be done in $2^{n-\Omega(n/\delta^5)}$ time [4]. The speedup is obtained by a fast modular counting algorithm for small prime powers and the Chinese remainder theorem after noting that the Hamiltonian cycles cannot be too many in a sparse graph.

1.1.4 Parity algorithms

In addition to the above decision algorithms, we know of an even faster algorithm for the seemingly more difficult problem of computing the parity of the number of Hamiltonian cycles: There is an $O(1.619^n)$ time algorithm computing the parity in directed graphs [2]. For bipartite graphs there is also an $O(1.5^n)$ time algorithm [2], and for bipartite undirected graphs, the $O(1.415^n)$ time decision algorithm in [1] is also capable of computing the parity. Thomason [16] showed that the parity of the number of Hamiltonian cycles through any specific edge in an undirected graph is always even in a graph where every vertex has odd degree. Note that it does not mean that there always are an even number of Hamiltonian cycles in the graph, e.g. K_4 has three Hamiltonian cycles. In fact, computing the parity in planar undirected graphs of maximum degree three is $\oplus$P-hard [18].

1.1.5 Sparsity-aware TSP algorithms

For the n-vertex Travelling Salesman Problem, i.e., in an edge-weighted graph find the Hamiltonian cycle of smallest total weight, there is a $2^{n-\Omega(n/2^{\Delta})}$ time algorithm, with Δ the maximum degree in the graph [3]. The proof uses the fact that in a dynamic programming across vertex subsets for Hamiltonian cycles one only needs to consider induced subgraphs that have degree at least two at every vertex. An upper bound of these can be found by the use of Shearer's lemma. There is also a $2^{n-\Omega(n/2^{2^{\delta}})}$ time algorithm with δ the average degree in the graph [7].

2 The Three Parts of our Design

On a high level, our algorithmic design consists of three parts:

1. Defining an algebraic fingerprint for Hamiltonicity with few non-zero terms in expectation.
2. Encoding possibly non-zero terms as solutions to a CNF-SAT formula.
3. Listing the solutions to the formula by a separate algorithm.

We will first describe each of these three parts before we present the algorithm in pseudocode in Section 3 along with its analysis.

2.1 A family of algebraic fingerprints

Let $G = (U, V, E)$ be a balanced $|U| = |V| = \frac{n}{2}$ bipartite undirected graph. We will introduce variables for the directed versions of the edges in G. We will next define a polynomial over a field of characteristic two as an exponential sum of determinants, and prove that it can be used to detect the presence of a Hamiltonian cycle in G. The construction and its analysis are very similar to the one for bipartite graphs in [1] with only one major difference. In [1] matrices in which rows and columns represented vertices from one part of the bipartition were used. Here we take an alternative approach with rows representing one part and columns representing the other part, as we feel it is more natural for describing how many terms can vanish in the summation in sparse graphs.

The idea is to define the polynomial so that it will be non-zero only if there is a Hamiltonian cycle in the graph. We will in fact describe an exponential number of exponential sums, all of which evaluate to the same polynomial. These are identified by variables $a_{i,j}$ for every $ij \in E$, and will not contribute to the sum. I.e., regardless of what they are set to, in the exponential sum they will cancel each other and the sum will always evaluate to the same value. They are introduced solely to make sure that under a random assignment, the expected number of non-zero terms in the exponential sum is quite small. We will show that in the next section.

Continuing the fingerprint design, we note that every Hamiltonian cycle $H \subseteq E$ can be oriented in two ways in the sense that starting from any vertex you can choose which of its two neighbors on the cycle to visit next. We will fix a special vertex $s \in V$ in the graph which we will use to break symmetry with respect to orientations. I.e., every oriented Hamiltonian cycle will be associated with a monomial in the polynomial, and the introduced asymmetry around s will ensure that different monomials are assigned to the two orientations of any Hamiltonian cycle to avoid that they cancel each other.

For every edge $ij \in E$ with $i \neq s$ and $j \neq s$, we introduce two identical variables $z_{i,j} = z_{j,i}$, i.e., they are only different names for the same variable, but for every $is \in E$ we introduce the two different variables $z_{i,s}$ and $z_{s,i}$. For $ij \notin E$, we set $z_{i,j} = z_{j,i} = 0$. We define a

polynomial matrix in a third set of variables $x = \{x_v | v \in V\}$, with rows representing vertices from V, and columns representing vertices from U, as

$$M_{i,j}(a,x,z) = \sum_{k \in V \setminus \{i\}} z_{i,j} z_{j,k} (a_{j,k} + x_k). \tag{1}$$

We will use

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

as a fingerprint of the existence of a Hamiltonian cycle in G. I.e., we sum over all $2^{n/2}$ assignments to x with each $x_i \in \{0,1\}$ to obtain a polynomial in the z-variables only (monomials with a-variables will cancel each other). We will prove that $\phi(G)$ can only be non-zero if G has a Hamiltonian cycle.

▶ **Lemma 3.** *Let $\mathcal{H}$ be the family of oriented Hamiltonian cycles in G, then*

$$\phi(G) = \sum_{h \in \mathcal{H}} \prod_{uv \in h} z_{u,v}. \tag{3}$$

Proof. Consider the Leibnitz expansion of the $n \times n$ matrix determinant in characteristic two,

$$\det(B) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} B_{i,\sigma(i)}.$$

If we furthermore expand each product of entries of the matrix $M(a,x,z)$ into a sum of products, we have that each term in the Leibnitz expansion of $\det(M(a,x,z))$ is the product of exactly $n/2$ factors, each of which is either $z_{i,j}z_{j,k}a_{j,k}$ or $z_{i,j}z_{j,k}x_k$ for some i,j,k with $i \neq k$. This means a term is the product of n z-variables and $n/2$ a- or x-variables. We first note that if such a term does not contain x_v for some $v \in V$, it will be counted an even number of times in (2). Let Z be the set of these omitted vertices v, and note that the term will be included both for $x_v = 0$ and $x_v = 1$ for all $v \in Z$ for any fixed assignment to the other variables. It will be counted $2^{|Z|}$ times, an even number for non-empty Z, and hence it will cancel in a characteristic two summation. This also means that in a surviving term, i.e., a term that is counted an odd number of times, each x_v for $v \in V$ is included precisely once as there are at most $n/2$ of them in total. Moreover, this means that no surviving term includes an a-variable.

Note that a term that includes x_v for all $v \in V$ describes a cycle cover as every double-arc factor is from a unique row (every vertex in V has outdegree 1), from a unique column (every vertex in U is the endpoint of exactly one arc and the start of another), and has a unique x-variable (every vertex in V has indegree 1). Moreover, there are no cycles on exactly 2 vertices due to the constraint $i \neq k$ in the summation in (1). Every cycle cover corresponds to some term in the Leibnitz expansion. If a cycle cover has more than one cycle, we can fix the lexicographically first cycle C that does not go through the special vertex s, and reverse its orientation to obtain a different cycle cover with the same monomial term. However this cycle cover is the result of another term in the Leibnitz expansion because the reversed cycle has length larger than 2. If we apply the cycle reversal operation again, the original cycle cover is obtained. Hence we have paired up all cycle covers with at least two cycles, proving that these will also cancel each other in a characteristic two summation. We are left with the Hamiltonian cycles, counted in both orientations as claimed. ◀

2.1.1 Limiting the number of contributing terms

The value of $\phi(G)$ in (3) is insensitive to the a-variables. No matter what we set them to the result is the same. We will now take advantage of this fact. By choosing a random assignment to the a-variables, we will end up with a formula in which many determinant terms for assignments to the x-variables in (2) will be trivially zero, and there is no need for us to evaluate them to compute $\phi(G)$.

We say that an assignment $x : V \to \{0, 1\}$ is *possibly contributing* if no column of $M(a, x, z)$ is all zeros. That is, $\det(M(a, x, z))$ is not trivially zero for the reason of having a column with no non-empty entries.

We will bound the probability that not too many assignments to x are possibly contributing under a random assignment to the variables a. Consider a fixed assignment x and let ε_u for $u \in U$ be the event that the column representing u in $M(a, x, z)$ is not identically zero under a randomly uniformly chosen a. We have from (1) that if $a_{u,v} + x_v = 0 (\mathrm{mod}\ 2)$ for all $uv \in E$ for a fixed u, then the event $\overline{\varepsilon}_u$ happens, hence

$$\Pr(\varepsilon_u) = 1 - \Pr\left(\prod_{uv \in E} (1 + a_{u,v} + x_v)) = 1 (\mathrm{mod}\ 2)\right) = 1 - \frac{1}{2^{d_u}},$$

where d_u is the degree of vertex u in G. Clearly, the events $\{\varepsilon_u : u \in U\}$ are mutually independent as they depend on different independent a-variables, so

$$\Pr\left(\bigcap_{u \in U} \varepsilon_u\right) = \prod_{u \in U} \left(1 - 2^{-d_u}\right). \tag{4}$$

By using Jensen's inequality for a concave function φ,

$$\varphi\left(\frac{\sum_{i=1}^{m} x_i}{m}\right) \geq \frac{\sum_{i=1}^{m} \varphi(x_i)}{m},$$

after noting that $\varphi(x) = \log(1 - 2^{-x})$ is concave, we have from (4) that

$$\Pr\left(\bigcap_{u \in U} \varepsilon_u\right) \leq (1 - 2^{-\delta})^{n/2}.$$

Consequently, by the linearity of expectation, the expected number of possibly contributing terms in (2) under a random assignment to the a-variables is at most

$$2^{n/2} \Pr\left(\bigcap_{u \in U} \varepsilon_u\right) \leq (2 - 2^{1-\delta})^{n/2}. \tag{5}$$

2.2 Encoding possibly contributing terms as a CNF SAT formula

We will next turn to how we can classify which x-assignments are possibly contributing without explicitly constructing all the matrices $M(a, x, z)$.

To encode the possibly contributing assignments, we consider propositional Boolean formulas in conjunctive normal form (CNF-SAT). An instance $\mathcal{I} = (W, \mathcal{C})$ consists of a set of variables W, and a set of clauses $\mathcal{C}$. Each clause in $\mathcal{C}$ is a finite set of literals, and a literal is an occurrence of a variable in W that may or may not be negated. For every assignment a we associate a CNF-SAT instance $\mathcal{I}(a) = (W_a, \mathcal{C}_a)$, where W_a is a set of $n/2$ Boolean variables,

with one variable w_v for each $v \in V$ with the interpretation that $w_v = \text{True} \iff x_v = 1$. Remember, we have from (1) that the event ε_u happens if some $a_{u,v} + x_v = 1 (\text{mod } 2)$ for some $uv \in E$. Let $P_u = \{v : uv \in E, a_{u,v} = 0\}$ and $N_u = \{v : uv \in E, a_{u,v} = 1\}$. Hence the clause

$$C_u = \left(\bigvee_{v \in P_u} w_v \right) \bigvee \left(\bigvee_{v \in N_u} \overline{w}_v \right),$$

where $\overline{w}_v$ means the negation of w_v, expresses the truth-value of event ε_u. We equate $\mathcal{C}_a$ with the set $\{C_u, u \in U\}$ as the clauses' conjunction expresses that all events happen. Consequently, every solution to $\mathcal{I}(a)$ represents an assignment $x : V \to \{0, 1\}$ that is possibly contributing. Note that $\mathcal{I}(a)$ has $n/2$ variables and $n/2$ clauses. We next turn to how to find them efficiently.

2.3 Listing possibly contributing terms

We will show that given a CNF-SAT instance $\mathcal{I}$ on ℓ variables and as many clauses, its solutions can be listed fast enough for our application.

▶ **Lemma 4.** *The solutions to a CNF-SAT formula on ℓ variables and at most ℓ clauses can be listed by a polynomial space algorithm in time*

$$O(1.619^\ell + s \operatorname{poly}(\ell)),$$

where s is the number of solutions.

Remark: Note that in our case $\ell = n/2$, so the first term amounts to a 1.272^n running time term that is dominated by the second term as there will be $(2 - 2^{1-\delta})^{n/2}$ solutions in expectation according to (5), which is larger than 1.272^n for $\delta \geq 3$.

Proof. The algorithm is a two-step procedure. First, as long as there is a variable that occurs both as positive and negative literals in the clauses and there are more than two occurrences of them, we use branching on that variable. Second, when no such variables exist, we can use an idea implicit in Tovey [17] to construct a bipartite perfect matching between clauses and variables to see if there is a solution at all. If so, we branch on any vertex and repeat to learn each variable's value one at a time for each of the assignments.

A partial assignment sets each variable in w to either True, False, or Undecided. Initially all variables are Undecided. We will gradually turn partial assignments into full assignments that satisfy the original instance. In the first step, we produce a set $\mathcal{S}$ of tuples of partial assignments and CNF-SAT instances resulting from the original instance by removing all clauses satisfied by the partial assignment. These will all have the following property: if a variable occurs both positively and negatively in the clauses, it has precisely two occurrences. The set $\mathcal{S}$ is generated by taking any yet undecided variable that occurs at least three times and both positively and negatively and setting it in turn to both truth values and recursively continuing on the clauses that are still unsatisfied by the partial assignment so far. If we let $t(\ell)$ be an upper bound on the number of instances generated this way from an original instance with ℓ clauses, we have that

$$t(\ell) \leq t(\ell - 1) + t(\ell - 2),$$

since at least one respectively two clauses are satisfied by the two assignments. In combination with $t(0) = 1$, we can solve this recurrence as $t(\ell) \leq 1.619^\ell$. Hence $|\mathcal{S}| \leq 1.619^\ell$.

In the second step, we consider each partial assignment and instance pair in $\mathcal{S}$ in turn. To see if the instance has a solution at all, we can first set all undecided variables that occur either only positively or negatively to the value that would satisfy some remaining clauses. After that we are left with a set of variables that occur in precisely two clauses but of opposite polarity. We construct a bipartite graph with one part representing clauses and one representing vertices, and edges between a clause and its variables. If such a bipartite graph has a perfect matching, we know the instance can be satisfied, by assigning to the variables the truth value that would satisfy the clause associated to the variable by the matching. Conversely, if there is a satisfying assignment, we can also find a perfect matching by connecting clauses with the variable that makes them true.

As long as there is a perfect matching, we know there is at least one satisfying assignment. We branch on any yet undecided variable and check again if there is a perfect matching. If not, we backtrack to another branch, but otherwise we continue to a full assignment and output it. This way we can list all satisfying assignments to the current instance in $\mathcal{S}$ with polynomial delay, as checking for a perfect matching is a polynomial time task. In fact, in our case checking for a perfect matching is particularly easy as every variable vertex on one side of the bipartition has only two choices. We can imagine another graph with vertices representing clauses, and variables representing edges, with edges between any two clauses that share a variable. Hall's marriage theorem now yields that a perfect matching in the original clause-variable bipartite graph exists if every connected component in the latter imagined graph has a cycle. This can be checked in linear time. ◀

2.4 An alternative listing algorithm for $\delta < 5.5$

If the average degree is small enough, we can use another way of listing the solutions, albeit using exponential space. We will use the path decomposition construction of Kneis et al. [13] to obtain a path-decomposition P_G of width at most $\mathrm{pw}(P_G) = \delta n/11.538$ in polynomial time. We first take the incidence graph of the CNF-SAT instance, the bipartite graph with vertices for clauses and variables, and edges between a clause and all its variables.

We can next use a path decomposition of the incidence graph to compute the number of satisfying assignments by a dynamic programming algorithm that uses one bit per variable vertex to keep track of its truth value, and one bit per clause vertex to keep track of whether the clause has been satisfied by the variables seen so far in the dynamic programming. The algorithm is analysed in Samer and Szeider [14] for tree-width, but leaving out the join nodes from the analysis gives a $2^{\mathrm{pw}(G)}\,\mathrm{poly}(n)$ time algorithm.

From the resulting dynamic programming table, we can list the solutions to the CNF-SAT formula with polynomial delay. Since the time needed to compute the dynamic programming across the path-decomposition is smaller than the expected number of solutions to the CNF-SAT instance

$$2^{\delta/11.538} < (2 - 2^{1-\delta})^{1/2},$$

for all $\delta \leq 5.5$, our running time bound follows.

We also note that Kneis et al. [13] presents another slightly larger path-width decomposition P'_G of size $\mathrm{pw}(P'_G) \leq \delta n/10.434$ that has a particularly nice structure: All bags share a large fraction of the vertices, and the remaining vertices induce a graph of constant bounded path-width. Hence one can get rid of the exponential space requirement with these path-decompositions for our application by guessing the assignment of the common vertices of all bags (i.e., try all of them), and then solve for the solutions by a path-decomposition dynamic programming of constant size in the remaining vertices. This works for all $\delta \leq 4.97$.

3 Algorithm

We are ready to describe and analyse the decision algorithm in pseudocode, and thereby prove Theorem 1. The algorithm operates over the field $\mathrm{GF}(2^\kappa)$. For now, it suffices to think of the parameter κ as logarithmic in n, it will be given a precise value in the next section. The following procedure is called n times, and as soon as a call returns "yes" we report the detection of a Hamiltonian cycle, otherwise we report no Hamiltonian cycles found.

HamiltonianCycle(G).

1. Pick random values $a : U \times V \to \{0,1\}$.
2. Use Lemma 4 to construct a list L of solutions to $\mathcal{I}(a)$.
3. If in the process more than $3(2-2^{1-\delta})^{n/2}$ solutions are found, abort immediately and return "no".
4. Set $s = 0$.
5. Pick random values $z : V \times U \to \mathrm{GF}(2^\kappa)$.
6. Set $z_{u,v} = z_{v,u}$ for $u \neq s$.
7. Pick random values $z : \{s\} \times V \to \mathrm{GF}(2^\kappa)$.
8. Set $z_{u,v} = 0$ for all $uv \neq E$.
9. For each $x \in L$,
10. Evaluate $t = \det(M(a,x,z))$ over $\mathrm{GF}(2^\kappa)$.
11. $s = s + t$ over $\mathrm{GF}(2^\kappa)$.
12. If $s \neq 0$ return "yes" otherwise return "no".

3.1 Analysis

We first analyse the correctness of the algorithm. We will set κ so that the probability of false negatives in a call to **HamiltonianCycle(G)** is at most $\frac{1}{2}$. By calling the procedure n times the claimed false negative probability in Theorem 1 follows.

Consider first step 2 of the algorithm. The expected number of solutions is

$$\mathbb{E}(|L|) = (2-2^{1-\delta})^{n/2},$$

according to (5). By Markov's inequality,

$$\Pr(|L| \geq 3 \cdot \mathbb{E}(|L|)) \leq \frac{1}{3}.$$

Hence the false negative rate reported at step 3 is at most $\frac{1}{3}$. If the algorithm proceeds past step 3, steps 4–12 computes (2), which according to Lemma 3 is non-zero only if G has a Hamiltonian cycle. Hence there is no chance of false positives. We use the following well-known Lemma to bound the probability of false negatives.

▶ **Lemma 5** (DeMillo–Lipton–Schwartz–Zippel[8, 15]). *Let $p(x_1, x_2, ..., x_m)$ be a nonzero m-variate polynomial of total degree d over a field F. Pick $r_1, r_2, ..., r_m \in F$ uniformly and independently at random, then*

$$\Pr(p(r_1, r_2, ..., r_m) = 0) \leq \frac{d}{|F|}.$$

In our case the polynomial has degree n as seen by (3) and because of the asymmetry around s it is a non-zero polynomial whenever there are Hamiltonian cycles in the graph. We use $F = \mathrm{GF}(2^\kappa)$ with $\kappa = \lceil \log 4n \rceil$ in the above Lemma, to get false negative rate at most $\frac{1}{4}$. Combining the two sources of false negatives, we get total false negative probability at most

$$\frac{1}{3} + \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{2},$$

as claimed.

We next analyse the running time. In step 2 we use the listing algorithm in Lemma 4 (or the alternative from 2.4) to list the solutions but aborting as soon as $3(2-2^{1-\delta})^{n/2}$ solutions have been found. According to the lemma, this takes $O(1.619^n + 3(2-2^{1-\delta})^{n/2}\,\mathrm{poly}(n))$ time, which is dominated by the second term for $\delta \geq 3$. Note that basic arithmetic computations over $\mathrm{GF}(2^\kappa)$ can be done in $\mathrm{polylog}(n)$ time. The determinant computation at step 10 requires polynomial time in n by using Gaussian elimination and multiplication of the diagonal elements to retrieve the value of the determinant (note that the sign of a permutation doesn't matter in characteristic two).

3.2 The proof of the parity counting theorem

We finally show how to modify the decision algorithm to obtain Theorem 2. First, to get a Las Vegas algorithm we simply omit bailing out in step 3 of the algorithm if the list of solutions to $\mathcal{I}(a)$ is too long. This gives a list of solution of expected length $(2-2^{1-\delta})^{n/2}$ according to (5). Second, we will replace the random values $z_{u,v}$ for $u, v \neq s$ by ones if $uv \in E$, and zeros otherwise. Third, we will run the algorithm several times, once for each pair of distinct neighbors v, w of s, setting only $z_{v,s} = z_{s,w} = 1$ whereas all other variables incident on s, $z_{s,u}$ and $z_{u,s}$ are set to zero for every u. This will make sure that we only count the parity of Hamiltonian cycles through v, s, w and in one orientation. Summing over all pairs of neighbors to s we obtain the parity of the number of all Hamiltonian cycles.

References

1. Andreas Björklund. Determinant Sums for Undirected Hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.
2. Andreas Björklund and Thore Husfeldt. The Parity of Directed Hamiltonian Cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 727–735, 2013. doi:10.1109/FOCS.2013.83.
3. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The traveling salesman problem in bounded degree graphs. *ACM Trans. Algorithms*, 8(2):18:1–18:13, 2012. doi:10.1145/2151171.2151181.
4. 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. doi:10.4230/LIPIcs.ICALP.2017.91.
5. Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity Checking Via Bases of Perfect Matchings. *J. ACM*, 65(3):12:1–12:46, 2018.
6. 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 *52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 150–159. IEEE Computer Society, 2011.
7. Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. *Inf. Comput.*, 243:75–85, 2015.

8 Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

9 David Eppstein. The Traveling Salesman Problem for Cubic Graphs. *J. Graph Algorithms Appl.*, 11(1):61–81, 2007.

10 Fedor V. Fomin and Kjartan Høie. Pathwidth of cubic graphs and exact algorithms. *Inf. Process. Lett.*, 97(5):191–196, 2006.

11 Kazuo Iwama and Takuya Nakashima. An Improved Exact Algorithm for Cubic Graph TSP. In *COCOON*, volume 4598 of *Lecture Notes in Computer Science*, pages 108–117. Springer, 2007.

12 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

13 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. A Bound on the Pathwidth of Sparse Graphs with Applications to Exact Algorithms. *SIAM J. Discrete Math.*, 23(1):407–427, 2009.

14 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

15 Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

16 Andrew G. Thomason. Hamiltonian Cycles and Uniquely Edge Colourable Graphs. In B. Bollobás, editor, *Advances in Graph Theory*, volume 3 of *Annals of Discrete Mathematics*, pages 259–268. Elsevier, 1978.

17 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

18 Leslie G. Valiant. Completeness for Parity Problems. In *COCOON*, volume 3595 of *Lecture Notes in Computer Science*, pages 1–8. Springer, 2005.