On the Exact Learnability of Graph Parameters: The Case of Partition Functions

Nadia Labai*[1] and Johann A. Makowsky†[2]

1   Department of Informatics, Vienna University of Technology, Vienna, Austria
     labai@forsyte.at
2   Department of Computer Science, Technion - Israel Institute of Technology,
     Haifa, Israel
     janos@cs.technion.ac.il

──────── **Abstract** ────────

We study the exact learnability of real valued graph parameters f which are known to be representable as partition functions which count the number of weighted homomorphisms into a graph H with vertex weights α and edge weights β. M. Freedman, L. Lovász and A. Schrijver have given a characterization of these graph parameters in terms of the k-connection matrices $C(f, k)$ of f. Our model of learnability is based on D. Angluin's model of exact learning using membership and equivalence queries. Given such a graph parameter f, the learner can ask for the values of f for graphs of their choice, and they can formulate hypotheses in terms of the connection matrices $C(f, k)$ of f. The teacher can accept the hypothesis as correct, or provide a counterexample consisting of a graph. Our main result shows that in this scenario, a very large class of partition functions, the rigid partition functions, can be learned in time polynomial in the size of H and the size of the largest counterexample in the Blum-Shub-Smale model of computation over the reals with unit cost.

1   Introduction

A graph *parameter* $f : \mathcal{G} \to \mathcal{R}$ is a function from all finite graphs $\mathcal{G}$ into a ring or field $\mathcal{R}$, which is invariant under graph isomorphisms.

In this paper we initiate the study of exact learnability of graph parameters with values in $\mathcal{R}$, which is assumed to be either $\mathbb{Z}, \mathbb{Q}$ or $\mathbb{R}$. As this question seems new, we focus here on the special case of graph parameters given as partition functions, [10, 14]. We adapt the model of exact learning introduced by D. Angluin [1]. Our research extends the work of [3, 11], where exact learnability of languages (set of words or labeled trees) recognizable by multiplicity automata (aka weighted automata) was studied, to graph parameters with values in $\mathcal{R}$.

1.1   Exact learning

In each step, the learner may make *membership queries* VALUE(x) in which they ask for the value of the target f on specific input x. This is the analogue of the MEMBERSHIP queries used in the original model of exact learning, [2]. The learner may also propose a hypothesis h by sending an EQUIVALENT(h) query to the teacher. If the hypothesis is correct, the teacher returns "YES" and if it is incorrect, the teacher returns a counterexample. A class of functions is *exactly learnable* if there is a learner that for each target function f, outputs a hypothesis h such that $f(x) = h(x)$ for all x and does so in time polynomial in the size of a shortest representation of f and the size of a largest counterexample returned by the teacher.

1.2   Formulating a hypothesis

To make sense one has to specify the formalism (language) $\mathfrak{L}$ in which a hypothesis has to be formulated. It will be obvious in the sequel, that the restriction imposed by the choice of $\mathfrak{L}$ will determine whether f is learnable or not.

Let us look at the seemingly simpler case of learning integer functions $f : \mathbb{Z} \to \mathbb{Z}$ or integer valued functions of words $w \in \Sigma^\star$ over an alphabet in Σ.

(i) If f can be any function $f : \mathbb{Z} \to \mathbb{Z}$ or $f : \Sigma^\star \to \mathbb{Z}$, there are uncountably many candidate functions as hypotheses, and no finitary formalism $\mathfrak{L}$ is suitable to formulate a hypothesis.

(ii) If f is known to be a polynomial $p(X) = \sum_i a_i X^i \in \mathbb{Z}[X]$, we can formulate the hypothesis as a vector $\mathbf{a} = (a_1, \ldots, a_m)$ in $\mathbb{Z}^m$. Learning is successful if the learner finds the hypothesis $h = \mathbf{a}$ in the required time. Here Lagrange interpolation will be used to formulate the hypotheses.

(iii) If f is known to satisfy some recurrence relation, the hypothesis will consist of the coefficients and the length of the recurrence relation, and exact learnability will depend on the class of recurrence relations one has in mind.

(iv) If $f : \Sigma^\star \to \mathbb{Z}$ is a word function recognizable by a multiplicity automaton MA, the hypotheses are given by the weighted transition tables of MA, cf. [3].

Looking now at a graph parameter $f : \mathcal{G} \to \mathcal{R}$ what can we expect? Again we have to restrict our treatment to a class of parameters where each member can be described by a finite string in a formalism $\mathfrak{L}$.

We illustrate the varying difficulty of the learning problem with the example of the chromatic polynomial $\chi(G; X \in \mathbb{N}[X])$ for a graph G. For $X = k$, the evaluation of $\chi(G; k)$ counts the number of proper colorings of G with at most k colors. It is well known that for fixed G, $\chi(G; k)$ is indeed a polynomial in k, [4, 7]. A graph parameter f is a *chromatic invariant over* $\mathcal{R}$ if

(i) it is multiplicative, i.e., for the disjoint union $G_1 \sqcup G_2$ of G_1 and G_2, it holds that $f(G_1 \sqcup G_2) = f(G_1) \cdot f(G_2)$, and

(ii) there are $\alpha, \beta, \gamma \in \mathcal{R}$ such that $f(G) = \alpha \cdot f(G_{-e}) + \beta \cdot f(G_{/e})$ and $f(K_1) = \gamma$.

K_n denotes the complete graph on n vertices, and G_{-e} and $G_{/e}$ are, respectively, the graphs obtained from deleting the edge e from G and contracting e in G.

The parameter $\chi(G; k)$ is a chromatic invariant with $\alpha = 1, \beta = -1$ and $\gamma = k$. Finally, $\chi(G; k)$ has an interpretation by counting homomorphisms:

$$\chi(G; m) = \sum_{t : G \to K_m} 1,$$
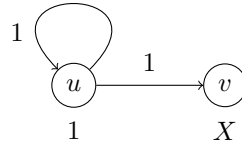
Figure 1 The weighted graph H_{indep}.

This is a special case of the homomorphism counting function for a fixed graph H:

$$\mathsf{hom}(G, H) = \sum_{t:G\to H} 1,$$

where t is a homomorphism $t : G \to H$.

Now, let a graph parameter $f : \mathcal{G} \to \mathcal{R}$ be the target of a learning algorithm.

(i) If f is known to be an instance of $\chi(G; X)$, a hypothesis consists of a value $X = a$. But in this case we know that $\chi(K_1; X) = X$, so it suffices to ask for $f(K_1) = a$.

(ii) If f is known to be a chromatic invariant, the hypothesis consists of the triple (α, β, γ). In this case a hypothesis can be computed from the values of $f(P_m)$ for undirected paths P_m for sufficiently many values of m.

(iii) If f is known to be an instance of $\mathsf{hom}(-, H)$, a hypothesis would consist of a target graph H.

1.3 Counting weighted homomorphisms aka partition functions

A *weighted graph* $H(\alpha, \beta)$ is a graph $H = (V(H), E(H))$ on $n = |V(H)|$ vertices together with a vertex weight function $\alpha : V(H) \to \mathbb{R}$, viewed as a vector of length n, and an edge weights function $\beta : V(H)^2 \to \mathbb{R}$ viewed as an $n \times n$ matrix, with $\beta(u, v) = 0$ if $(u, v) \notin E(H)$.

A *partition function*[1] $\mathsf{hom}(-, H(\alpha, \beta))$ is the generalization of $\mathsf{hom}(-, H)$ to weighted graphs, whose value on a graph G is defined as follows:

$$\mathsf{hom}(G, H(\alpha, \beta)) = \sum_{t:G\to H} \prod_{v\in V(G)} \alpha(t(v)) \prod_{(u,v)\in V(G)^2} \beta(t(u), t(v))$$

To illustrate the notion of a partition function, let H_{indep} be the graph with two vertices $\{u, v\}$ and the edges $\{(u, v), (u, u)\}$, shown in Figure 1. Let $\alpha(u) = 1, \alpha(v) = X$ and $\beta(u, v) = 1, \beta(u, u) = 1$. Then $\mathsf{hom}(-, H_{indep}(\alpha, \beta))$ is the *independence polynomial*,

$$\mathsf{hom}(G, H_{indep}(\alpha, \beta)) = I(G; X) = \sum_j ind_j(G) X^j$$

where $ind_j(G)$ is the number of independent sets of size j in the graph G.

We say a partition function $\mathsf{hom}(-, H(\alpha, \beta))$ is *rigid* aka *asymmetric* [2], if H has no proper automorphisms. Note that automorphisms in a weighted graph also respect vertex and edge weights. In our examples above, the evaluations of the independence polynomial are rigid partition functions, whereas the evaluations of the chromatic polynomial are not. It is known that almost all graphs are rigid:

---

[1] In the literature $\mathsf{hom}(-, H(\alpha, \beta))$ is also denoted by $Z_{H(\alpha,\beta)}(G)$, e.g., in [19]. We follow the notation of [14].

[2] Some authors say G is asymmetric if G has no proper automorphisms, and G is rigid if G has no proper endomorphisms, [12]. Wikipedia uses rigid as we use it here.

Algorithm 1 Learning algorithm for rigid partition functions

1: $n = 1$
2: **while** True **do**
3:      augment M with(B_n)
4:      $P = $ find basis(M)
5:      $h = $ generate hypothesis(P)
6:      **if** EQUIVALENT$(h) = $ YES **then**
7:          **return** h
8:      **else**
9:          $n = n + 1$
10:         $B_n = $ EQUIVALENT(h) ▷ B_n receives a counterexample
11:     **end if**
12: **end while**

▶ **Theorem 1** ([9, 12]). *Let G be a uniformly selected graph on n vertices. The probability that G is rigid tends to 1 as $n \to \infty$.*

If the target f is known to be a (rigid) partition function $\mathsf{hom}(-, H(\alpha, \beta))$ then the hypothesis consists of a (rigid) weighted graph $H(\alpha, \beta)$.

In Section 2 we give the characterization of rigid and non-rigid partition functions from [10, 15, 14] in terms of connection matrices.

For technical reasons discussed in Section 5, in this paper we deal only with the learnability of rigid partition functions, and leave the general case to future work.

1.4   Main result

Our main result can now be stated:

▶ **Theorem 2.** *Let f be a graph parameter which is known to be a rigid partition function $f(G) = \mathsf{hom}(G, H(\alpha, \beta))$. Then f can be learned in time polynomial in the size of H and the size of the largest counterexample in the Blum-Shub-Smale model of computation over the reals with unit cost.*

▶ **Remark 3.** *If f takes values in $\mathbb{Q}$ rather than in $\mathbb{R}$ we can also work in the Turing model of computation with logarithmic cost for the elements in $\mathbb{Q}$.*

To prove Theorem 2 we will use the characterization of rigid partition functions in terms of connection matrices, [14, Theorem 5.54], stated as Theorem 4 and Corollary 6 in Section 2. The difficulty of our result lies not in finding a learning algorithm by carefully manipulating the counterexamples to meet the complexity constraints, but in proving the algorithm correct. In order to do this we had to identify and extract the suitable algebraic properties underlying the proof of Theorem 4 and Corollary 6.

The learning algorithm is given in pseudo-code as Algorithm 1. It maintains a matrix M used in the generation of the hypothesis h from VALUE and EQUIVALENT query results. After an initial setup of M, in each iteration the algorithm generates a hypothesis h, queries the teacher for equivalence between h and the target and either terminates, or updates M accordingly and moves on to the next iteration.

It uses three black-boxes; find basis which uses M to find a certain basis P of a graph algebra associated with the target function (see Section 2), generate hypothesis which uses

this basis and VALUE queries to construct a hypothesis h, and augment M which augments the matrix M after a counterexample is received, using VALUE queries.

We briefly overview the complexity of the algorithm to illustrate that rigid partition functions are indeed exactly learnable. Proofs of validity and detailed analysis of the complexity are given in later sections. For a target $H(\alpha, \beta)$ on q vertices, the procedure find basis solves $O(q)$ systems of linear equations, and systems of linear matrix equations, all of dimension $O(\text{poly}(q))$. The procedure generate hypothesis performs $O(q)$ graph operations of polynomial time complexity on graphs of size $O(\text{poly}(q, |x|))$, where $|x|$ is the size of the largest counterexample, and $O(q^2)$ VALUE queries. The procedure augment M performs $O(q)$ VALUE queries. Thus, each iteration takes time $O(\text{poly}(q, |x|))$. Lemma 18 will show that there are $O(q)$ iterations, so the total run time of the algorithm is polynomial in the size q of $H(\alpha, \beta)$ and the size $|x|$ of the largest counterexample.

Organization

In Section 2 we give the necessary background on partition functions and the graph algebras induced by them. Section 3 presents the algorithm in detail and in Section 4 we prove its validity and analyze its time complexity. We discuss the results and future work in Section 5. Due to space limitations, the appendix is included in the arXiv version, [13].

2 Preliminaries

Let $k \in \mathbb{N}$. A k-*labeled graph* G is a finite graph in which k vertices, or less, are labeled with labels from $[k] = \{1, \ldots, k\}$. We denote the class of k-labeled graphs by $\mathcal{G}_k$. The k-*connection* of two k-labeled graphs $G_1, G_2 \in \mathcal{G}_k$ is given by taking the disjoint union of G_1 and G_2 and identifying vertices with the same label. This produces a k-labeled graph $G = G_1 G_2$. Note that k-connections are commutative.

2.1 Quantum graphs

A formal linear combination of a finite number of k-labeled graphs $F_i \in \mathcal{G}_k$ with coefficients from $\mathbb{R}$ is called a k-*labeled quantum graph*. $\mathcal{Q}_k$ denotes the set of k-labeled quantum graphs.

Let x, y be k-labeled quantum graphs: $x = \sum_{i=1}^n a_i F_i$, and $y = \sum_{i=1}^n b_i F_i$. Note that some of the coefficients may be zero. $\mathcal{Q}_k$ is an infinite dimensional vector space, with the operations: $x + y = (\sum_{i=1}^n a_i F_i) + (\sum_{i=1}^n b_i F_i) = \sum_{i=1}^n (a_i + b_i) F_i$, and $\alpha \cdot x = \sum_{i=1}^n (\alpha a_i) F_i$.

k-connections extend to k-labeled quantum graphs by $xy = \sum_{i,j=1}^n (a_i b_j)(F_i F_j)$. Any graph parameter f extends to k-labeled quantum graphs linearly: $f(x) = \sum_{i=1}^n a_i f(F_i)$.

2.2 Equivalence relations for quantum graphs

The k-*connection matrix* $C(f, k)$ of a graph parameter $f : \mathcal{G} \to \mathcal{R}$ is a bi-infinite matrix over $\mathcal{R}$ whose rows and columns are labeled with k-labeled graphs, and its entry at the row labeled with G_1 and the column labeled with G_2 contains the value of f on $G_1 G_2$:

$$C(f, k)_{G_1, G_2} = f(G_1 G_2).$$

Given a connection matrix $C(f, k)$, we associate with a k-labeled graph $G \in \mathcal{G}_k$ the (infinite) row vector R_G^k appearing in the row labeled by G in $C(f, k)$. If k is clear from context we write R_G. Similarly, we associate an infinite row vector R_x with k-labeled quantum graphs

$x = \sum_{i=1}^{n} a_i F_i$, defined as $R_x = \sum_{i=1}^{n} a_i R_{F_i}$ where R_{F_i} is the row in $C(f, k)$ labeled by the k-labeled graph F_i.

We say $C(f, k)$ has *finite rank* if there are finitely many k-labeled graphs $\mathcal{B}_{C(f,k)} = \{B_1, \ldots, B_n\}$ whose rows $\mathcal{R}_{C(f,k)} = \{R_{B_1}, \ldots, R_{B_n}\}$ linearly span $C(f, k)$. Meaning, for any k-labeled graph G, there exists a linear combination of the rows in $\mathcal{R}_{C(f,k)}$ which equals the row vector R_G. We say that $C(f, k)$ has rank n and denote $r(f, k) = n$ if any set of less than n graphs does not linearly span $C(f, k)$.

The main result we use is the characterization of partition functions in terms of connection matrices. We do not need its complete power, so we state the relevant part:

▶ **Theorem 4** (Freedman, Lovász, Schrijver, [10]). *Let f be a graph parameter that is equal to* $\mathsf{hom}(-, H(\alpha, \beta))$ *for some $H(\alpha, \beta)$ on q vertices. Then $r(f, k) \leq q^k$ for all $k \geq 0$.*

The exact rank $r(f, k)$ was characterized in [15], but first we need some definitions. A weighted graph $H(\alpha, \beta)$ is said to be *twin-free* if β does not contain two separate rows that are identical to each other [3]. Let $H(\alpha, \beta)$ be a weighted graph on q vertices, and let $\mathrm{Aut}(H(\alpha, \beta))$ be the automorphism group of $H(\alpha, \beta)$. $\mathrm{Aut}(H(\alpha, \beta))$ acts on ordered k-tuples of vertices $[q]^k = \{\phi : [k] \to [q]\}$ by $(\sigma \circ \phi)(i) = \sigma(\phi(i))$ for $\sigma \in \mathrm{Aut}(H(\alpha, \beta))$. The *orbit* of ϕ is the set of ordered k-tuples ψ of vertices such that $\sigma \circ \phi = \psi$ for an automorphism $\sigma \in \mathrm{Aut}(H(\alpha, \beta))$. The *number of orbits* of $\mathrm{Aut}(H(\alpha, \beta))$ on $[q]^k$ is the number of different orbits for elements $\phi \in [q]^k$.

▶ **Theorem 5** (Lovász, [15]). *Let $f = \mathsf{hom}(-, H(\alpha, \beta))$ for a twin-free weighted graph $H(\alpha, \beta)$ on q vertices. Then $r(f, k)$ is equal to the number of orbits of $\mathrm{Aut}(H(\alpha, \beta))$ on $[q]^k$ for all $k \geq 0$.*

We use the special case:

▶ **Corollary 6.** *Let $f = \mathsf{hom}(-, H(\alpha, \beta))$ for a rigid twin-free weighted graph $H(\alpha, \beta)$ on q vertices. Then $r(f, k) = q^k$ for all $k \geq 0$.*

We define an equivalence relation $\equiv_{f,k}$ over $\mathcal{Q}_k$ where two k-labeled quantum graphs x and y are in the same equivalence class if and only if the infinite vectors R_x and R_y are identical: $x \equiv_{f,k} y \iff R_x^k = R_y^k$. Note that the set $\mathcal{Q}_k/f$ of equivalence classes of $\equiv_{f,k}$ is exactly the vector space $span(C(f, k))$ generated by linear combinations of rows in $C(f, k)$. k-connections extend to these vectors by: $R_x R_y = R_{xy}$.

Thus, if $r(f, k) = n$ with spanning rows $\mathcal{R}_{C(f,k)} = \{R_{B_1}, \ldots, R_{B_n}\}$, they form a basis of $\mathcal{Q}_k/f = span(C(f, k))$. For brevity, we occasionally also refer to $\mathcal{B}_{C(f,k)}$ as a basis.

Let x be a k-labeled quantum graph whose equivalence class R_x is given as the linear combination $R_x = \sum_{i=1}^{n} \gamma_i R_{B_i}$. We call the column vector $\bar{c}_x = (\gamma_1, \ldots, \gamma_n)^T$ the *coefficients vector* of x, or *representation* of x using $\mathcal{B}_{C(f,k)}$.

3   The learning algorithm in detail

In this section we present the learning algorithm in full detail. The commentary in this exposition foreshadows the arguments in Section 4, but otherwise validity is not considered here. We do not address complexity concerns in this section either, however, we reiterate

---

[3] If $H(\alpha, \beta)$ has twin vertices, they can be merged into one vertex by adding their vertex weights without changing the partition function. As the size of the target representation is the smallest possible, we assume all targets are twin-free.

for the sake of clarity that the algorithm runs on a Blum-Shub-Smale machine, [6, 5], over the reals. In such a machine, real numbers are treated as atomic objects; they are stored in single cells, and arithmetic operations are performed on them in a single step.

The objects the algorithm primarily works with are real matrices. In a context containing a basis $\mathcal{B}_{C(f,k)}$, we associate a real matrix A_x with each quantum graph x such that the following holds.

The coefficients vector $\bar{c}_{xy}$ of xy using $\mathcal{B}_{C(f,k)}$ is given by $A_x\bar{c}_y$. $\qquad\qquad\qquad$ (*)

This device, as we will see in Section 4, will allow the algorithm to search for, and find, special quantum graphs that provide a translation of the answers of VALUE and EQUIVALENT queries into a hypothesis.

As mentioned earlier, Algorithm 1 maintains a matrix M which is a submatrix of $C(f, 1)$. In each iteration the algorithm generates a hypothesis $h = (\alpha^{(h)}, \beta^{(h)})$ using M, and queries the teacher for equivalence between h and the target f. If the hypothesis is correct, the algorithm returns h, otherwise it augments M with a 1-labeled version of the counterexample, and moves on to the next iteration.

▶ **Remark 7.** *Strictly speaking, the teacher may be asked* VALUE *queries on (unlabeled) graphs, however, we freely write* VALUE(G) *for k-labeled graphs $G \in \mathcal{G}_k$. Additionally, the algorithm will need to know the value of the target on some quantum graphs. Since any graph parameter extend to quantum graphs linearly, for a quantum graph $x = \sum_{i=1}^{n} a_i F_i$ we write* VALUE(x) *as shorthand for $\sum_{i=1}^{n} a_i \cdot$ VALUE(F_i) throughout the presentation.*

Incorporating counterexamples

The objective is to keep a non-singular submatrix M of $C(f, 1)$. The first 1-labeled graph B_1 with which M is augmented is some arbitrarily chosen 1-labeled graph.

Upon receiving a B_n graph as counterexample, the 1-label is arbitrarily assigned to one of its vertices, making it a 1-labeled graph. Then augment M with(B_n) adds a row and a column to M labeled with the (now) 1-labeled graph B_n, and fills their entries with the values $f(B_nB_i) = f(B_iB_n)$, for $i \in [n]$, using VALUE queries.

The other functions are slightly more complex.

Finding an idempotent basis

The function find basis, given in pseudo-code as Algorithm 2, receives as input the matrix M. For reasons which will become apparent later, we are interested in finding a certain (idempotent) basis of the linear space generated by the rows of $C(f, 1)$. For this purpose, in its first part find basis iteratively, over $k = 1, \ldots, n$, computes the entries of matrices A_x as in (*), where x are B_i, $i \in [n]$, by solving multiple systems $M\mathbf{x} = \mathbf{b}$ of linear equations, and using the solutions Γ of those systems to fill the entries of the matrices A_{B_i}, where the (k, j) entry of A_{B_i} is $\boldsymbol{\gamma}^{ij}(k)$. Let p_i, $i \in [n]$ be those quantum graphs for which A_{p_i} is the $n \times n$ matrix with the value 1 in the entry (i, i) and zero in all other entries. Note that the matrices A_{p_i}, $i \in [n]$ are linearly independent. We will see that p_i, $i \in [n]$ are the idempotent basis, now we wish to find their representation using B_i, $i \in [n]$.

For $i \in [n]$, the representation $\bar{c}_{p_i}$ of the basic idempotent p_i using the basis elements $B_i, i \in [n]$ is found by solving a system $A\mathbf{X} = A_{p_i}$ of linear *matrix equations*, where A is a block matrix whose blocks are the matrices A_{B_i}, $i \in [n]$. Each solution is added to Δ.

Finally, find basis outputs the set Δ of these representations $\bar{c}_{p_i}$, $i \in [n]$. Then we have that $R_{p_i} = \sum_{k=1}^{n} \bar{c}_{p_i}(k)R_{B_k}$ where $\bar{c}_{p_i}$ is the coefficients vector of p_i using B_i, $i \in [n]$. The

---

Algorithm 2 find basis function

1: $\Gamma = \emptyset$
2: **for** each $i, j \in [n]$ **do**
3:      **for** $k = 1, \ldots, n$ **do**
4:          $\mathbf{b}(k) = \text{VALUE}(B_i B_j B_k)$
5:      **end for**
6:      $\boldsymbol{\gamma}^{ij} = $ solve linear system$(M\mathbf{x} = \mathbf{b})$
7:      $\Gamma = \Gamma \cup \{\boldsymbol{\gamma}^{ij}\}$
8: **end for**
9: **for** $i \in [n]$ **do**
10:     $A_{B_i} = $ fill matrix(i, Γ)
11:     $A = $ add block(A, i, A_{B_i}) $\triangleright$ A is a block matrix with A_{B_i} on its ith block
12: **end for**
13: $\Delta = \emptyset$
14: **for** $i \in [n]$ **do**
15:     $\bar{c}_{p_i} = $ solve linear matrix system$(A\mathbf{X} = A_{p_i})$
16:     $\Delta = \Delta \cup \{\bar{c}_{p_i}\}$
17: **end for**
18: **return** Δ

---

representations $\bar{c}_{p_i} \in \Delta$ of the elements p_i, $i \in [n]$, are what will provide a translation from results of VALUE queries to weights.

Generating a hypothesis

The function generate hypothesis, given in pseudo-code as Algorithm 3, receives as input the representations $\bar{c}_{p_i}$ of the 1-labeled quantum graphs p_i, $i \in [n]$, which it uses to find the entries of the vertex weights vector $\alpha^{(h)}$ directly through VALUE queries.

Then generate hypothesis finds the 2-labeled analogues of these 1-labeled quantum graphs. Those 2-labeled analogues form a basis of of $\mathcal{Q}_2/f$.

Denote by K_2 the 2-labeled graph composed of a single edge with both vertices labeled. Next, generate hypothesis finds the representation of $R^2_{K_2}$, that is the row labeled with K_2 in $C(f, 2)$, using the basis $\mathcal{R}_{C(f,2)}$. We find the representation of this specific graph K_2 as the coefficients in $\bar{c}_{K_2}$ constitute the entries of the edge weights matrix $\beta^{(h)}$ (see Section 4).

This representation is found by solving a linear system of equations, similarly to how find basis uses solve linear system, but here we use the diagonal matrix N whose entries correspond to the elements of $\mathcal{B}_{C(f,2)}$.

The solution of said system, i.e., the coefficients vector $\bar{c}_{K_2}$ of K_2, is used to fill the edge weights matrix $\beta^{(h)}$. If needed, $\beta^{(h)}$ is made twin-free by contracting the twin vertices into one and summing their weights in $\alpha^{(h)}$.

Finally, generate hypothesis returns the hypothesis $h = (\alpha^{(h)}, \beta^{(h)})$ as output.

▶ **Remark 8** (Algorithm 3). *Let q_i be the 1-labeled quantum graph p_i interpreted as a 2-labeled quantum graph, and let q_j be p_j with the labels of its components renamed to 2, and also interpreted as a 2-labeled quantum graph. The result of $p_i \otimes p_j$ is the 2-labeled quantum graph $q_i \sqcup_2 q_j$.*

Algorithm 3 generate hypothesis function

1: **for** each $i \in [n]$ **do**
2:      $\alpha^{(h)}(i) = \text{VALUE}(p_i)$
3: **end for**
4: $N = 0^{n^2 \times n^2}$ $\triangleright$ N is a zero matrix of dimensions $n^2 \times n^2$.
5: **for** $i = 1, \dots, n$ **do**
6:      **for** $j = 1, \dots, n$ **do**
7:          $p_{ij} = p_i \otimes p_j$ $\triangleright$ See Remark 8.
8:          $N_{p_{ij}, p_{ij}} = \text{VALUE}(p_{ij} p_{ij})$
9:          $\mathbf{b}(i_j) = \text{VALUE}(K_2 \, p_{ij})$
10:      **end for**
11: **end for**
12: $\beta^{(h)} = \text{solve linear system}(N\mathbf{x} = \mathbf{b})$
13: make twin-free$(\alpha^{(h)}, \beta^{(h)})$
14: $h = (\alpha^{(h)}, \beta^{(h)})$
15: **return** h

---

4   **Validity and complexity**

As stated earlier, a class of functions is exactly learnable if there is a learner that for each target function f, outputs a hypothesis h such that f and h identify on all inputs, and does so in time polynomial in the size of a shortest representation of f and the size of a largest counterexample returned by the teacher. The proof of Theorem 2 argues that Algorithm 1 is such a learner for the class of rigid partition functions, through Theorem 9, which proves validity, and Theorem 22, which proves the complexity constraints are met.

To prove validity, we first state existing results on properties of graph algebras induced by partition functions, then show, through somewhat technical algebraic manipulations, how our algorithm successfully exploits these properties to generate hypotheses. We then show our algorithm eventually terminates with a correct hypothesis.

For the rest of the section, let $H(\alpha, \beta)$ be a rigid twin-free weighted graph on q vertices, and denote $f = \text{hom}(-, H(\alpha, \beta))$.

▶ **Theorem 9.** *Given access to a teacher for f, Algorithm 1 outputs a hypothesis h such that $f(G) = h(G)$ for all graphs $G \in \mathcal{G}$.*

The proof of the theorem follows from arguing that:

▶ **Theorem 10.** *If M is of rank q, then generate hypothesis outputs a correct hypothesis.*

and that the rank of M is incremented with every counterexample:

▶ **Theorem 11.** *In the nth iteration of Algorithm 1 on f, M has rank n.*

First we confirm the hypotheses Algorithm 1 generates are indeed in the class of graph parameters we are trying to learn, namely, rigid partition functions $\text{hom}(-, H(\alpha, \beta))$ for twin-free weighted graphs $H(\alpha, \beta)$.

Given Theorem 11, for the hypothesis h returned in the nth iteration, the rank of $C(h, 1)$ is at least n, since M is a submatrix of $C(h, 1)$. Thus, from Theorem 5, h cannot have proper automorphisms, as it would imply that the rank of $C(h, 1) < n$. The fact that h is twin-free is immediate from the construction in generate hypothesis.

4.1   From the idempotent bases to the weights – proof of Theorem 10

Let $\mathcal{Q}_k/f$ be of finite dimension n. The *idempotent basis* $p_1, \ldots, p_n$ of $\mathcal{Q}_k/f$ consists of those k-labeled quantum graphs p_i for which $p_i p_i \equiv_{f,k} p_i$ and $p_i p_j \equiv_{f,k} 0$ for $i, j \in [n]$, $i \neq j$. Recall how find basis found those 1-labeled quantum graphs p_i, $i \in [n]$ whose matrices A_{p_i} behaved in this way.

In our setting of rigid twin-free weighted graphs, by [14, Chapter 6], we have that if $p_1, \ldots, p_q$ are the idempotent basis of Q_1/f, then the idempotent basis of $\mathcal{Q}_2/f$ is given by $p_i \otimes p_j$, $i, j \in [q]$. These are the 2-labeled analogues mentioned in the description of generate hypothesis.

Furthermore by [14, Chapter 6], the vertex weights α of H are given by $\alpha(i) = f(p_i)$, $i \in [q]$, and if the representation of K_2 using $p_i \otimes p_j$, $i, j \in [q]$ is $\sum_{i,j \in [q]} \beta_{ij}(p_i \otimes p_j)$, then the edge weights matrix β is given by $\beta_{i,j} = \beta_{ij}$.

Equipped with these useful facts, we show that:

▶ **Lemma 12.** *If M is of rank q, then* find basis *outputs the idempotent basis of $\mathcal{Q}_1/f$.*

Then obtain Theorem 10 by showing how, if generate hypothesis receives the idempotent basis of $\mathcal{Q}_1/f$ as input, it outputs a correct hypothesis.

Finding the idempotent basis – proof of Lemma 12

Recall that in the presence of a basis $\mathcal{B}_{C(f,k)}$ we associate a real matrix A_x with each quantum graph x such that the following holds.

$$\text{The coefficients vector } \bar{c}_{xy} \text{ of } xy \text{ using } \mathcal{B}_{C(f,k)} \text{ is given by } A_x \bar{c}_y.$$

For reasons we cannot list here, A_x will be diagonal. Let $B_i, B_j \in \mathcal{B}_{C(f,1)}$, and denote by $\sum_{k=1}^n \gamma_k^{i,j} R_{B_k}$ the representation of the row $R_{B_i B_j}$ using $\mathcal{R}_{C(f,1)}$, i.e., the row in $C(f,1)$ labeled with the graph resulting from the product $B_i B_j$.

▶ **Claim 13.** *Let x be some 1-labeled quantum graph such that $R_x = \sum_{i=1}^n a_i R_{B_i}$. The matrix A_x is given by $(A_x)_{\ell,m} = \sum_{i=1}^n a_i \gamma_\ell^{im}$.*

Note that for a basis graph $B_k \in \mathcal{B}_{C(f,1)}$, we have that $(A_{B_k})_{i,j} = \gamma_i^{k,j}$. The proof of this claim appears in the appendix of [13].

▶ **Proposition 14.** *The matrices $A_{B_1}, \ldots, A_{B_n}$ of the graphs in $\mathcal{B}_{C(f,1)}$ are linearly independent and span all matrices of the form A_x for a quantum graph x.*

If we know what are the matrices $A_{p_1}, \ldots, A_{p_n}$ of the idempotent basis $p_1, \ldots, p_n$, we can find their representation using $A_{B_1}, \ldots, A_{B_n}$ by solving systems of linear matrix equations. Then, given a representation $A_{p_i} = \sum_{k=1}^n \delta_k^{(i)} A_{B_k}$, we will have the representation of the basic idempotents using $\mathcal{B}_{C(f,1)}$ as $p_i = \sum_{k=1}^n \delta_k^{(i)} B_k$.

The definitions of A_x and idempotence lead to the observation that for idempotent basics p_i, p_j, it holds that $A_{p_i} A_{p_i} = A_{p_i}$ and $A_{p_i} A_{p_j} = 0$. From Corollary 6 we know the dimension of $\mathcal{Q}_1/f$ is q, so we conclude:

▶ **Proposition 15.** *The idempotent basis for $\mathcal{Q}_1/f$ consists of the quantum graphs p_i, $i \in [q]$ for which A_{p_i} is the $q \times q$ matrix with the value 1 in the entry (i,i) and zero in all other entries. That is,*

$$A_{p_i}(k,j) = \begin{cases} 1, & \text{if } (k,j) = (i,i) \\ 0, & \text{otherwise} \end{cases}$$

As find basis solves the systems of linear matrix equations for these matrices, it remains to show that find basis correctly computes the matrices A_{B_i}, $i \in [q]$.

Since M is of full rank, the representations $\sum_{k=1}^{n} \gamma_k^{i,j} R_{B_k}$ of graphs $B_i B_j$, $i,j \in [q]$ using $\mathcal{B}_{C(f,1)}$ are correctly computed by the solve linear system calls. And as noted before, the coefficients $\gamma_k^{i,j}$ are the entries of the matrices A_{B_i}, $i \in [q]$. Thus they indeed are correctly computed, and we have Lemma 12.

Since generate hypothesis directly queries the teacher for the values of $\alpha^{(h)}$, we have:

▶ **Corollary 16.** *If M is of rank q, then generate hypothesis outputs a correct vertex weights vector $\alpha^{(h)}$.*

It remains to show this is true also for the edge weights:

▶ **Proposition 17.** *If M is of rank q, then generate hypothesis outputs a correct edge weights matrix $\beta^{(h)}$.*

Proof. As $p_{ij} = p_i \otimes p_j$, $i,j \in [q]$ are the idempotent basis for $\mathcal{Q}_2/f$ we have that $p_{ij} p_{ij} \not\equiv_{f,2} 0$, so the matrix N is a diagonal matrix of full rank, and solve linear system indeed finds the representation of K_2 using p_{ij}, $i,j \in [q]$. ◀

From Corollary 16 and Proposition 17 we have Theorem 10.

Now we show that Algorithm 1 reaches that point in the first place.

4.2 Augmentation results in larger rank – proof of Theorem 11

Theorem 11 is proved using the fact that A_x are linearly independent for k-labeled quantum graphs which are not equivalent in $\equiv_{f,k}$.

▶ **Lemma 18.** *In the nth iteration of Algorithm 1, if the teacher returns a counterexample x, then R_x is not spanned by $R_{B_1}, \ldots, R_{B_n}$ where $B_1, \ldots, B_n$ are the graphs associated with the rows and columns of M.*

Proof. If $n = 1$, M has rank n. Now let M have rank n.

For contradiction, assume that $R_x = \sum_{i=1}^{n} a_i R_{B_i}$. Then $x \equiv_{f,1} \sum_{i=1}^{n} a_i B_i$ and we have that $\mathsf{hom}(x, H) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i, H)$ for the target graph H. Denote by $h^{(n)}$ the hypothesis generated in this iteration. If x is a counterexample, it must hold that

$$\mathsf{hom}(x, h^{(n)}) \neq \mathsf{hom}(x, H) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i, H)$$

The solution of the system of equations for $\mathbf{b}_x$ would give

$$\mathsf{hom}(x, h^{(n)}) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i, h^{(n)}) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i, H)$$

So we conclude that $\sum_{i=1}^{n} a_i \mathsf{hom}(B_i, h^{(n)}) \neq \sum_{i=1}^{n} a_i \mathsf{hom}(B_i, H)$.

Since M is of full rank, one can solve a system of linear equations using M for $\mathbf{b}_x$ defined as $\mathbf{b}_x(k) = \text{VALUE}(xB_k)$, $k \in [n]$. Now recall that the matrix M contains *correct* values $\mathsf{hom}(B_i B_j, H(\alpha, \beta))$, as it was augmented using VALUE queries, therefore M is a submatrix of $C(f,1)$. Thus the coefficients of the solution $\mathbf{a}$ of $M\mathbf{a} = \mathbf{b}_x$ equal a_i, $i \in [k]$, and we reach a contradiction. Therefore we conclude $x \not\equiv_{f,1} \sum_{i=1}^{n} a_i B_i$ and its row R_x is linearly independent from $R_{B_1}, \ldots, R_{B_n}$. ◀

This also implies that the matrix A_x associated with x is not spanned by $A_{B_1}, \ldots, A_{B_n}$. Therefore the submatrix of $C(f,1)$ composed of the entries of the rows and columns of $B_1, \ldots, B_n, x$ is of full rank $n+1$. This is exactly the matrix M augmented with x, and we have Theorem 11. Combining this with Corollary 6, we have:

▶ **Corollary 19.** *Let f be a rigid partition function of a twin-free weighted graph on q vertices. Then Algorithm 1 terminates in q iterations.*

4.3   Complexity analysis

As the algorithm runs on a Blum-Shub-Smale machine for the reals and mostly solves systems of linear equations, it is not difficult to show that it runs in time polynomial in the size of target and the largest counterexample. First we observe:

▶ **Proposition 20.** *Let $G_1, G_2 \in \mathcal{G}_1$. Then $G_1 G_2$ can be computed in time $O(\mathrm{poly}(|G_1|, |G_2|))$.*

▶ **Remark 21.** *B_1 is of fixed size, and all other B_i, $i = 2, \ldots, n$, used in Algorithm 1 are counterexamples provided by the teacher, therefore they are all of size polynomial in the size $|x|$ of the graph x.*

▶ **Theorem 22.** *Let $H(\alpha, \beta)$ be a rigid twin-free weighted graph on q vertices and denote $f = \mathsf{hom}(-, H(\alpha, \beta))$. Given access to a teacher for f, Algorithm 1 terminates in time $O(\mathrm{poly}(q, |x|))$, where $|x|$ is the size of the largest counterexample provided by the teacher.*

Proof. From Corollary 19 it is enough to show that each iteration of Algorithm 1 does not take too long (Lemma 23). ◀

▶ **Lemma 23** ([13]). *In the nth iteration of Algorithm 1, $\mathsf{augment}$ M, $\mathsf{find\ basis}$, and $\mathsf{generate\ hypothesis}$ all run in time $O(\mathrm{poly}(n, |x|))$.*

▶ **Remark.** We note that, from [14, Theorem 6.45], the counterexamples provided by the teacher may be chosen to be of size at most $2(1 + q^2)q^6$ where q is the size of the target weighted graph.

5   Conclusion and future work

This paper presented an adaptation of the exact model of learning of Angluin, [1], to the context of graph parameters f representable as partition functions of weighted graphs $H(\alpha, \beta)$. We presented an exact learning algorithm for the class of *rigid* partition functions defined by twin-free $H(\alpha, \beta)$.

If a weighted graph has proper automorphisms, its connection matrices $C(f, k)$ may have rank smaller than q^k. In this case, the translation from query results to a weighted graph would involve the construction of a submatrix of $C(f, k)$ for a sufficiently large k, and then find an idempotent basis for $\mathcal{Q}_{k+1}/f$. We will study the learnability of non-rigid partition functions in a sequel to this paper.

Theorems similar to Theorem 4 have been proved for variants of partition functions and connection matrices, [16, 8, 17, 18]. It seems reasonable to us that similar exact learning algorithms exist for these settings, but it is unclear how to modify our proofs here for this purpose.

─── **References** ───

1 D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

2 D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

3 A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM (JACM)*, 47(3):506–530, 2000.

4 G. D. Birkhoff. A determinant formula for the number of ways of coloring a map. *Annals of Mathematics*, 14:42–46, 1912.

5 L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer Science & Business Media, 2012.

6 L. Blum, M. Shub, S. Smale, et al. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21(1):1–46, 1989.

7 B. Bollobás. *Modern Graph Theory*. Springer, 1999.

8 J. Draisma, D. C. Gijswijt, L. Lovász, G. Regts, and A. Schrijver. Characterizing partition functions of the vertex model. *Journal of Algebra*, 350(1):197–206, 2012.

9 P. Erdős and A. Rényi. Asymmetric graphs. *Acta Mathematica Hungarica*, 14(3-4):295–315, 1963.

10 M. Freedman, L. Lovász, and A. Schrijver. Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society*, 20(1):37–51, 2007.

11 A. Habrard and J. Oncina. Learning multiplicity tree automata. In *Grammatical Inference: Algorithms and Applications*, pages 268–280. Springer, 2006.

12 J. Kötters. Almost all graphs are rigid—revisited. *Discrete Mathematics*, 309(17):5420–5424, 2009.

13 N. Labai and J. A. Makowsky. On the exact learnability of graph parameters: The case of partition functions. *arXiv preprint arXiv:1606.04056*, 2016. URL: `http://arxiv.org/abs/1606.04056`.

14 L. Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. AMS, 2012.

15 L. Lovász. The rank of connection matrices and the dimension of graph algebras. *European Journal of Combinatorics*, 27(6):962–970, 2006.

16 A. Schrijver. Graph invariants in the spin model. *J. Comb. Theory, Ser. B*, 99(2):502–511, 2009.

17 A. Schrijver. Characterizing partition functions of the spin model by rank growth. *Indagationes Mathematicae*, 24.4:1018–1023, 2013.

18 A. Schrijver. Characterizing partition functions of the edge-coloring model by rank growth. *Journal of Combinatorial Theory, Series A*, 136:164–173, 2015.

19 A. D. Sokal. The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In *Survey in Combinatorics, 2005*, volume 327 of *London Mathematical Society Lecture Notes*, pages 173–226, 2005.