Not all Strongly Rayleigh Distributions Have Small Probabilistic Generating Circuits

Markus Bläser [1]

Abstract

Probabilistic modeling is a central task in machine learning. Probabilistic models should be tractable, i.e., allowing tractable probabilistic inference, but also efficient, i.e., being able to represent a large set of probability distributions. Zhang et al. (ICML 2021) recently proposed a new model, probabilistic generating circuits. They raised the question whether every strongly Rayleigh distribution can be efficiently represented by such circuits. We prove that this question has a negative answer. There are strongly Rayleigh distributions that cannot be represented by polynomial-sized probabilistic generating circuits, assuming a widely accepted complexity theoretic conjecture.

1. Introduction

Probabilistic modeling is a central task in machine learning. However, probabilistic inference easily becomes intractable when the underlying models become large and complicated, see (Roth, 1996) for a theoretical explanation. Therefore, it is important to develop probabilistic models that are tractable (TPMs for short), that is, they allow for probabilistic inference that is tractable in the size of the model. On the other hand, the probabilistic models should be as expressive efficient as possible (in the sense of Martens & Medabalimi (2014)), which means that they can represent as many classes of distributions as possible while staying small in size. The more classes we can represent, the broader the spectrum of applications of the model. However, there is typically a tradeoff between expressiveness and tractability. The more expressive the model, the harder will be probabilistic inference.

Examples of tractable models are for instance bounded treewidth graphical models (Meila & Jordan, 2000; Koller & Friedman, 2009), the well-known determininantal point processes (Borodin & Rains, 2005; Kulesza & Taskar, 2012), or probabilistic circuits like for instance sum-product networks (Darwiche, 2009; Kisa et al., 2014; Poon & Domingos, 2012). These models represent probability distributions by computing probability mass functions: the input is an assignment to the random variables and the output is the corresponding probability of the event.

Zhang et al. (2021) recently proposed a new model, probabilistic generating circuits (PCGs). Unlike probablistic circuits, they represent probability distributions by probability generating functions. Probability generating functions represent the joint distribution of a set of binary random variables as the coefficients of a multilinear polynomial. This new model is still tractable as it supports marginal inference. Furthermore it is very efficient, it subsumes all the probablistic models mentioned above in this regard.

What makes PGCs particularly attractive is the fact that many interesting probability distributions are defined via generating polynomials. Strongly Rayleigh (SR) distributions (Borcea et al., 2009) are a prominent example here. Zhang et al. (2021) leave it as a question for further research which classes of interesting distributions can we represented efficiently by PGCs. In particular, they ask the question whether all SR distributions can be expressed by a polynomial size PGC.

1.1. Our work

As our main result, we prove that this question has a negative answer, as has already been conjectured by Zhang et al. We prove that there is a family of SR distributions that does not have polynomial size PGCs assuming a widely believed complexity theoretic assumption, namely, that the polynomial time hierarchy is infinite.

1.2. Previous work

A distribution is strongly Rayleigh, if the corresponding generating polynomial is real stable. There is a natural candidate for a counter example to the question by Zhang et al., namely, matching polynomials. However, it turns out that they are not directly suited as a counter example, as we

[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany. Correspondence to: Markus Bläser <mblaeser@cs.uni-saarland.de>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

will explain now.

The matching polynomial of a graph $G = (V, E)$ with n nodes is defined as $\mu_G(x) = \sum_{i=0}^{\lceil n/2 \rceil} (-1)^i m_i x^{2i}$, where m_i is the number of matchings of size i of G, see Section 4 for definitions. This polynomial has a multivariate generalization, namely

$$M_G(x_1, \dots, x_n) = \sum_{M \text{ matching of } G} (-1)^{|M|} \prod_{i \text{ matched in } M} x_i.$$

Here, we sum over all matchings and for each matching, we add up one monomial ("label") that enumerates all matched nodes. M_G specializes to μ_G by replacing every x_i by the same variable x, that is $\mu_G(x) = M_G(x, \dots, x)$.

Assume that n is even. It is well-known that counting all perfect matchings, that is, computing $m_{n/2}$ is $\#\mathsf{P}$-complete (and in particular NP-hard) even for bipartite graphs, see Section 7 for definitions. It is a classical result that matching polynomials (univariate and multivariate) are real stable (Heilmann & Lieb, 1972; Amini, 2019). However, this does not prove that there are real stable polynomials that have large arithmetic circuits. In fact, the matching polynomial μ_G is univariate and therefore trivially has a small arithmetic circuit. Also, it is not clear whether M_G has small a circuit for a particular G. The $\#\mathsf{P}$-hardness of matchings only tells us that the mapping that maps a graph to its matching polynomial is hard, but *not* each polynomial itself.

One can define a more general matching polynomial that subsumes all matching polynomials: Let $e_{i,j}$, $1 \le i < j \le n$ be a set of $\binom{n}{2}$ variables and let K_n denote the complete graph on n nodes, where we identify the nodes with the set $\{1, \dots, n\}$. Let

$$M_n(x, e) = \sum_{M \text{ matching in } K_n} (-1)^{|M|} \prod_{\{i,j\} \in M} x_i e_{i,j} x_j. \quad (1)$$

This polynomials lists for each matching not only the variables corresponding to the nodes matched, but also a variable for each edge in the matching. Let $G = (V, E)$ be a graph on n nodes and let $\eta \in \{0, 1\}^{\binom{n}{2}}$ be the characteristic vector of E, that is, there is a 1 in the vector if the corresponding edge is in E and a 0 otherwise. Then we have

$$M_n(x, \eta) = M_G(x),$$

since a product in (1) survives iff all $e_{i,j}$ in it are set to 1. Therefore, M_n contains *every* matching polynomial and hence, in some sense, also the computational hardness of the mapping that maps a graph to its matching polynomial.

However, this does not solve the problem, as we were are not able to prove that this polynomial M_n is real stable. (For such reasons that simple polynomials like $1 + x^2$ or $1 - x^3$ are not real stable. More general, Choe et al. (2004) prove that the support of a real stable polynomial necessarily forms a jump system, as introduced by Bouchet & Cunningham (1995).) Therefore, we define a new and even more general multilinear polynomial that in the end turns out to be real stable and for which we can rule out that it has polynomial size arithmetic circuits under standard complexity theoretic assumptions.

1.3. Organisation of the paper

In Section 2 we introduce probability generating polynomials and circuits (PGC). We recall the definition of strongly Rayleigh (SR) distributions in Section 3. In Section 4, we introduce matchings and generalizations of it, which form the basis of our construction of a SR distribution that does not have small PGCs. In Section 5, we give the actual construction of our generating polynomial and prove in Section 6 that it is real stable. Section 7 contains basic definitions and facts from complexity theory that we need to prove our impossibility result. We show that the polynomials we defined are hard to evaluate (in a complexity theoretic sense) in Section 8. While the polynomial we constructed is real stable, it does not yet define a SR distribution. In Section 9, we modify the construction such that it becomes a probability generating polynomial while still staying real stable. Finally, we prove that this probability generating polynomial does not have polynomial size PGCs unless the polynomial time hierarchy collapses in Section 10. Our main result is Theorem 10.1 and its Corollary 10.2. Thus it is considered to be unlikely that these polynomial size PGCs exist, since it is a standard assumption in complexity theory that the polynomial time hierachy is infinite. Some proofs are deferred to Appendix A due to space limitations.

2. Probabilistic generating circuits

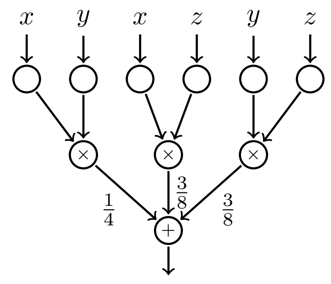
Generating polynomials are a tool in combinatorics used to encode sequences of numbers. In particular, probability distributions over binary random variables can be represented by multilinear polynomials.

Definition 2.1. Let P be some probability distribution over binary random variables $X_1, \dots, X_n$. The probability generating polynomial g for P is defined by

$$g(z_1, \dots, z_n) = \sum_{S \subseteq \{0,1\}} \alpha_S z^S,$$

where $\alpha_S = P(\{X_i = 1 : i \in S\}, \{X_i = 0 : i \notin S\})$ and $z^S = \prod_{i \in S} z_i$.

A probability generating polynomial represents each joint probability by the cofficient of the corresponding monomial. Probability generating polynomials have an exponential number of coefficients. Arithmetic circuits are a way to store them efficiently.



x	y	z	Pr
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	$\frac{3}{8}$
1	0	0	0
1	0	1	$\frac{3}{8}$
1	1	0	$\frac{1}{4}$
1	1	1	0

Figure 1. A probabilistic generating circuit representing the polynomial $\frac{1}{4}xy + \frac{3}{8}xz + \frac{3}{8}yz$, which is real stable. On the righthand side is the corresponding distribution, which is strongly Rayleigh.

Definition 2.2 ((Zhang et al., 2021))**.** A probabilistic generating circuit[1] (PGC) is a directed acyclic graph consisting of three types of nodes:

1. Nodes labelled with "+" (sum nodes), the edges going into sum nodes can be labelled with rational constants.
2. Nodes labelled with "$\times$" (product nodes)
3. Nodes of indegree 0 (leaves), which are labelled with constants or variables.

There is a unique node of outdegree 0, the output node.

Figure 1 shows a small PGC. PGCs are equivalent to arithmetic circuits, a standard model in complexity theory (Saptharishi, R. et al., 2021). Each node of a PGC represents a polynomial in the natural way: An input node represents a single variable polynomial or a constant. A sum node represents the weighed sum of the polynomials represented by its children and a product node the product of the polynomials represented by its children. The polynomial represented by the output node is the polynomial represented by the PGC.

The size of a PGC is the number of nodes in it, the description size is the length of some encoding of the PGC as a binary string. It is more natural to study the description size of a circuit, since it also takes into account the bit size of the constants involved. This avoids having degenerate cases in which the circuit has only a few nodes but constants with a large binary expansion.

Note that the coefficients of a PGC do not necessarily form a probability distribution. They do not need to sum up to 1 and can even be negative. The learning process has to ensure that we get a probability distribution in the end, see (Zhang et al., 2021) for a further discussion and examples.

[1]The term probabilistic generating circuit was coined by Zhang et al, however, it might be more natural to use the term *probability* generating circuit, since they compute probability generating polynomials.

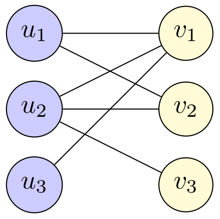


Figure 2. A bipartite graph with bipartition $U = \{u_1, u_2, u_3\}$ and $V = \{v_1, v_2, v_3\}$.

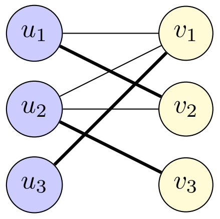


Figure 3. The thick edges are a perfect matching.

3. Strongly Rayleigh distributions

An important class of distributions are the strongly Rayleigh distributions (Borcea et al., 2009), which were first defined for studying negative dependence:

Definition 3.1. A polynomial $f \in \mathbb{R}[z_1, \dots, z_n]$ is real stable if $f(\zeta_1, \dots, \zeta_n) \neq 0$ for all $\zeta_1, \dots, \zeta_n \in \mathbb{C}$ with $\Im(\zeta_i) > 0$, $1 \leq i \leq n$. ($\Im$ denotes the imaginary part.)

Definition 3.2. A distribution on variables $X_1, \dots, X_n$ is strongly Rayleigh (SR), if its generating polynomial is real stable.

SR distributions are naturally characterized by their generating polynomials rather than by probability mass functions. Zhang et al. (2021) identify subclasses of SR distributions, like spanning tree distributions or determinantal point processes, that can be efficiently represented by PGCs. It is natural to ask whether all SR distributions can be efficiently represented by PGCs. We give a negative answer to this question.

4. Graphs, matchings, and functions

A graph G is called bipartite, if we can partition its nodes into two set U and V such that all edges have one node in U and the other node in V. When we write $G = (U \cup V, E)$ we mean that G is a bipartite graph with bipartition U and V. We will typically call these nodes $u_1, \dots, u_m$ and $v_1, \dots, v_n$ in the following. $M \subseteq E$ is called a matching if each node of U and V appears in at most one edge of M. M is called perfect, if every node is in exactly one edge. If a bipartite graph has a perfect matching, then necessarily $|U| = |V|$. The size $|M|$ of a matching is the number of edges in it. If M is perfect, then $|M| = |U| = |V|$.

Example 4.1. Figure 2 shows a bipartite graph with three nodes on each side. The thick edges in Figure 3 form a perfect matching. Figure 4 shows a matching of size two.

We can interpret a perfect matching M as a bijective function $U \to V$. If $e = \{u_i, v_j\}$ is an edge in M, then M maps u_i to v_j. If we identify u_i with i and v_j with j, then M can also be interpreted as a permutation of the number

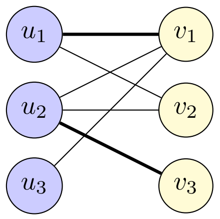


Figure 4. The thick edges are a matching of size two.

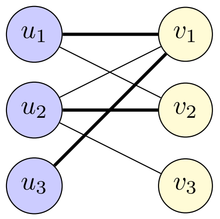


Figure 5. The thick edges form a total function $U \to V$, which is not injective.

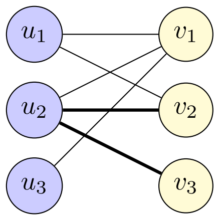


Figure 6. The thick edges form a partial function from V to U.

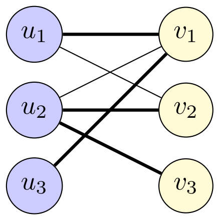


Figure 7. A double function.

$\{1,\dots,n\}$. In general, if M is a matching (but not necessarily perfect), then M defines a partial function which is injective when restricted to its domain.

Example 4.2. The perfect matching in Figure 3 defines the function that maps $u_1 \mapsto v_2$, $u_2 \mapsto v_3$, and $u_3 \mapsto v_1$. Interpreted as a permutation, it is the cyclic shift (123). The matching in Figure 4 maps $u_1 \mapsto v_1$ and $u_2 \mapsto v_3$ and is undefined on u_3. Restricted to its domain, it is an injective function. We can interpret it as a partial permutation mapping $1 \mapsto 1$ and $2 \mapsto 3$.

We can interpret a matching as a function that is injective when restricted to its domain. It turns out that to obtain our main result it is useful to consider arbitrary functions and not just injective ones.

Definition 4.3. Let $G = (U \cup V, E)$ be a bipartite graph. $F \subseteq E$ is called a (partial) function, if every node in U is incident to at most one edge in F.

Such a set F is called a function, because it naturally defines a partial function $U \to V$. For each node u, there is at most one edge $(u, v) \in F$.[2] Then v is the image of u. We write $F(u) = v$. If there is no such edge, then the function is undefined on u. The image of F is $\operatorname{im} F := \{v \in V \mid \exists\, u \in U : (u, v) \in F\}$, that is, all the nodes that appear on the righthand side of an edge in F. The domain of F, $\operatorname{dom} F := \{u \in U \mid \exists\, v \in V : (u, v) \in F\}$, is the set of all nodes in U on which F is defined. A function F is called total if $\operatorname{dom} F = U$.

Example 4.4. The thick edges in Figure 5 are the function mapping $u_1 \mapsto v_1$, $u_2 \mapsto v_2$, and $u_3 \mapsto v_1$. This function is total (all nodes in U are matched), but not injective, since v_1 is contained in two edges.

So far, we considered functions F from $U \to V$. We can also consider functions $V \to U$. A set $H \subseteq E$ is a (partial) function from $V \to U$ if each node in V is incident to at most one edge in H. For each node v, there is at most one edge $(u, v) \in H$. Then u is the image of v. We write $H(v) = u$.

[2]We will write edges directed from now, since later on, we will identify U with V.

Example 4.5. Figure 6 shows a function from V to U that maps $v_2 \mapsto u_2$ and $v_3 \mapsto u_2$. It is partial and not injective when restricted to its domain.

The next definition will be crucial for our construction.

Definition 4.6. Let $G = (U \cup V, E)$ be a bipartite graph.

1. $E' \subseteq E$ is called a (partial) double function, if $E' = F \cup H$ for a (partial) function $F : U \to V$ and a (partial) function $H : V \to U$.
2. E' is called a total double function, if in addition, F and H are both total functions.

Example 4.7. Figure 7 shows the double function that is created by combining the functions from Figure 5 and 6. It is not a total double function since the function in Figure 6 is not. The edge $\{u_2, v_2\}$ appears in both functions, but it only appears once in the double function. This will be an issue, because we cannot distinguish this from the case when the edge was missing in one of the two functions. Therefore, we will consider multigraphs in the following.

Such a graph is a bipartite graph with multiedges, that is, there might be several edges between two nodes u and v. Any of the edges between u and v can appear in a function that maps u to v. This will be important in the constructions that follow.

5. Function generating polynomials

Let $K_{m,n}$ denote the complete bipartite graph with m nodes on lefthand and n nodes on the righthand side, respectively. Let $K^{(d)}_{m,n}$ be the complete bipartite multigraph with m nodes on lefthand and n nodes on the righthand side and between each pair of nodes, there are exactly d copies of an edge. Obviously, $K_{m,n} = K^{(1)}_{m,n}$. Every bipartite multigraph with m and n nodes on each side, respectively, and maximum degree $\leq d$ can be obtained from $K^{(d)}_{m,n}$ by the deletion of edges. In this way, we will encode many graphs into one.

Example 5.1. Figure 8 shows the complete bipartite multigraph with three nodes on each side and three copies of each edge.

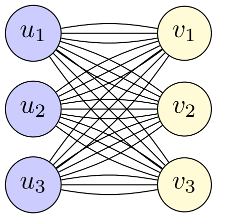


Figure 8. The complete bipartite multigraph $K_{3,3}^{(3)}$.

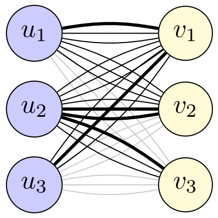


Figure 9. When we remove the light grey edges from $K_{3,3}^{(3)}$, we get the graph from Figure 7 with each edge triplicated. The thick edges form an instantiation of the double function in Figure 7.

We consider $K_{m,n}$. We will label the nodes on the lefthand and righthand side by $\{1,\dots,m\}$ and by $\{1,\dots,n\}$, respectively. This is slightly abusive, since these sets are formally not disjoint. However, we will write the edge set with a direction, that is, as $\{1,\dots,m\} \times \{1,\dots,n\}$, so there is no danger of confusing nodes on the lefthand side with ones on the righthand side. Let $e = (e_{i,j} : 1 \le i \le m, 1 \le j \le n\}$ be a set of variables, which we identify with the edges in the obvious way.

The signed double function generating polynomial is defined as

$$DF_{m,n}(e) = \sum_{F,H} (-1)^{|F|+|H|} \prod_{(i,j)\in F} e_{i,j} \prod_{(i',j')\in H} e_{i',j'}. \tag{2}$$

where the sum is taken over all subsets F of the edges of $K_{m,n}$ that are a partial function $U \to V$ and subsets H that are a partial function $V \to U$. For each double function of $K_{m,n}$, that is, each pair (F,H), the polynomial contains one monomial. This monomial lists the variables corresponding to edges that are contained in F and H. The sign factor in the front is important for obtaining real stability (see the next section). Note that if an edge appears in F and in H, then the monomial contains the corresponding variable with degree two. This turns out to be a problem in the interpolation process in the proof of Theorem 8.3. By forbidding variables of degree two, the weights that different double functions get in the proof of Theorem 8.3 depend on the number of shared edges. In this way, we are able to extract the number of perfect matchings in the end. Therefore, we generalize the double function generating polynomial to multigraphs and change the way how double functions are represented by monomials slightly but in an important way.

Example 5.2. The double function consisting of the two functions in Figures 5 and 6 corresponds to the monomial $e_{1,1}e_{2,2}^2e_{2,3}e_{3,1}$.

The signed double function generating polynomial generalizes to $K_{m,n}^{(d)}$ in the obvious way. Let $e^{(d)} = (e_{i,j}^{(\delta)} : 1 \le i \le m, 1 \le j \le n, 1 \le \delta \le d)$. We sum over all pairs of functions (F,H). For each edge, we now have d choices, so every double function gives rise to many monomials by choosing different copies. The definition of this generalization $DF_{m,n}^{(d)}$ is given in Equation (3) in Figure 10 (since it does not fit into one column). Note that when F and H share an edge e, then we are forced to take a different copy of e for F and H. This is an important difference that will circumvent the problem mentioned above for $DF_{m,n}$.

Example 5.3. Figure 9 shows an instantiation of the double function in Figure 7. The corresponding monomial is $e_{1,1}^{(1)}e_{2,2}^{(2)}e_{2,2}^{(3)}e_{2,3}^{(1)}e_{3,1}^{(2)}$. (The indices of the copies of the edges in Figure 9 are labelled from top to bottom.)

Note that we obtain $DF_{m,n}^{(d-1)}$ from $DF_{m,n}^{(d)}$ by simply setting all variables $e_{i,j}^{(d)}$ to zero. In the same way, we can define the signed double function generating polynomial for any bipartite multigraph G with degree bounded by d by zeroing out variables $e_{i,j}^{(t+1)},\dots,e_{i,j}^{(d)}$ when the edge (i,j) has multiplicity t in G.

We define the signed total double function generating polynomials $DFtot_{m,n}$ and $DFtot_{m,n}^{(d)}$ in the same way as $DF_{m,n}$ and $DF_{m,n}^{(d)}$. Instead of summing over all pair of functions (F,H), we sum over all pairs of total functions instead in Equations (2) and (3).

From the double function in Example 5.3, we cannot exactly recover the two functions forming the double function unambigously. One function (from $V \to U$) surely consists of $(2,2)$ and $(2,3)$ and the other (from $U \to V$) of $(2,2)$ and $(3,1)$. The edge $(1,1)$ can be put into any of the two functions. This means that the coefficient of the monomial in $DF_{m,n}$ will be 2, since the monomial can be created in two ways in the definition of $DF_{m,n}$. The same holds true for all instantiations of the double function in $DF_{m,n}^{(d)}$.

We can make it unambiguous by also denoting the domains of the two functions with every monomial. We take additional variables $u = (u_i : 1 \le i \le m)$ and $v = (v_j : 1 \le j \le n)$. The extended double function generating polynomials is defined in (4) in Figure 10. If the node i is in $\operatorname{dom} F$, then u_i appears in the corresponding monomial. If j is in $\operatorname{dom} H$, then v_j appears. In this way, each monomial does not contain the edges but also the domains of the two functions.

Example 5.4. The two monomials of $DFext_{3,3}^{(3)}$ corresponding to the monomial $e_{1,1}^{(1)}e_{2,2}^{(2)}e_{2,2}^{(3)}e_{2,3}^{(2)}e_{3,1}^{(2)}$ in Example 5.3 are $u_1u_2u_3e_{1,1}^{(1)}e_{2,2}^{(2)}e_{2,2}^{(3)}e_{2,3}^{(1)}e_{3,1}^{(2)}v_2v_3$ and $u_2u_3e_{1,1}^{(1)}e_{2,2}^{(2)}e_{2,2}^{(3)}e_{2,3}^{(1)}e_{3,1}^{(2)}v_1v_2v_3$.

The introduction of the variables $u_1,\dots,u_m$ and $v_1,\dots,v_n$ will not make the situation completely unambiguous. The

$$DF^{(d)}_{m,n}(e^{(d)}) = \sum_{F,H}(-1)^{|F|+|H|} \prod_{(i,j)\in F\setminus H} \sum_{\delta=1}^{d} e^{(\delta)}_{i,j} \prod_{(i',j')\in H\cap F} \sum_{1\le\delta'<\gamma\le d} e^{(\delta')}_{i',j'} e^{(\gamma)}_{i',j'} \prod_{(i'',j'')\in H\setminus F} \sum_{\delta''=1}^{d} e^{(\delta'')}_{i'',j''} \tag{3}$$

$$DFext^{(d)}_{m,n}(e^{(d)}, u, v) = \sum_{F,H}(-1)^{|F|+|H|} \prod_{(i,j)\in F\setminus H} \sum_{\delta=1}^{d} u_i e^{(\delta)}_{i,j} \prod_{(i',j')\in H\cap F} \sum_{1\le\delta'<\gamma\le d} u_{i'} e^{(\delta')}_{i',j'} e^{(\gamma)}_{i',j'} v^{(\gamma)}_{j'} \prod_{(i'',j'')\in H\setminus F} \sum_{\delta''=1}^{d} e^{(\delta'')}_{i'',j''} v_{j''} \tag{4}$$

Figure 10. The definitions of $DF^{(d)}_{m,n}$ and $DFext^{(d)}_{m,n}$. The sums are over all double functions (F,H) of $K_{m,n}$. The expressions in equations (3) and (4) generate all possible double functions of the complete bipartite multigraph on $m+n$ nodes. For each of them, they generate one monomial. The outer sum sums only over double functions of the complete bipartite graph and then in the inner sums, we choose the copies of the edges accordingly. To obtain the monomial of Example 5.3 we take as F the function $(1,1)$, $(2,2)$, $(3,1)$ (see Figure 5) and as H the function $(2,2)$, $(2,3)$ (see Figure 6). In the inner part, we now generate all monomials of the form $e^{(a)}_{1,1}e^{(b)}_{2,2}e^{(c)}_{2,2}e^{(d)}_{2,3}e^{(f)}_{3,1}$ with a,b,c,d,f being between 1 and d and $b<c$. This includes the monomial of Example 5.3. The first product in (3) runs over the edges $(1,1)$ and $(3,1)$, which are in F but not in H. The second product runs over the edge $(2,2)$, since it appears in F and H. The last product is over the remaining edge $(2,3)$. There is a second way to generate this monomial, as it also can arise from a different double function, see the discussion before Example 5.4.

important point is that we will use these variables to prove the real stability of $DFext^{(d)}_{m,n}$. Then, we will use $DFext^{(d)}_{m,n}$ to prove the real stability of $DF^{(d)}_{m,n}$.

Example 5.5. When F and H are both perfect matchings, then (F,H) and (H,F) will be mapped to the same monomials in $DFext^{(d)}_{m,n}$.

6. Double function generating polynomials are stable

The following polynomial will be the basic building block. It is well known and easy to see that it is real stable.

Lemma 6.1. $1-u_ie^{(\delta)}_{i,j}$ *is real stable.*

Lemma 6.2. $\prod_{1\le i,j\le n}\prod_{\delta=1}^{d}(1-u_ie^{(\delta)}_{i,j})$ *is real stable.*

Proof. The product of real stable polynomials is real stable, see e.g. (Pemantle, 2012). □

The polynomial above is a product of sums. In each factor, we can either choose the summand 1 or the summand $-u_ie^{(\delta)}_{i,j}$. Therefore, each monomial corresponds to a subset of the edges (with multiplicities). For each edge (i,j) and copy δ, we also multiply with the variable u_i, which represents the source of the edge. Therefore, we have the following lemma.

Lemma 6.3. *The monomials in* $\prod_{1\le i,j\le n}\prod_{\delta=1}^{d}(1-u_ie^{(\delta)}_{i,j})$ *that are multilinear in* $u_1,\dots,u_n$ *stand in one-to-one correspondence with partial funcions* $F: U\to V$ *and a vector* $(\delta_i)_{i\in\operatorname{dom}F}$, *specifying which copy of the edge* (i,j) *is taken.*

In the same way, we can consider the functions $V\to U$ by looking at the polynomial

$$\prod_{1\le i,j\le n}\prod_{\delta=1}^{d}(1-e^{(\delta)}_{i,j}v_j) \tag{5}$$

This polynomial is real stable, too, and Lemma 6.3 holds accordingly.

Lemma 6.4. *The multilinear monomials of the product of the two polynomials in Lemma 6.3 and Equation (5) stand in one-to-one correspondence with partial double functions* (F,H) *and vectors* $(\delta_i)_{i\in\operatorname{dom}F}$ *and vector* $(\gamma_j)_{j\in\operatorname{dom}H}$ *such that whenever* $(i,j)\in F\cap H$, *then* $\delta_i\neq\gamma_j$.

Proof. The multilinear monomials contain each u_i at most once. Therefore, at most one node on the lefthand side is matched. The same is true for the righthand side, since each v_j has degree at most one, too. Since double functions in $K^{(d)}_{m,n}$ need to choose different copies of an edge, if it appears in both functions F and H, the corresponding monomial is multilinear, too. □

The multi-affine part operator MAP takes a polynomial and removes all monomials that are not multilinear, that is, that contain at least one variable of degree two or higher. If x is a variable, the operator MAP_x takes a polynomial and removes all monomials that are not multilinear in x, that is, that contain x with a degree two or higher. We can write

$$\mathrm{MAP}_x(f) = (1+\hat{x}\partial_x)f|_{x=0}$$

where $\hat{x}$ is a new variable, that replaces x (and can be renamed into x afterwards). Here ∂_x denotes the partial derivative operator. MAP_x naturally generalizes to sets of variables. It is well-known that the MAP_x- and MAP-operator preserve real stability (Borcea & Brändén, 2009).

Lemma 6.5. *We have*

$$DFext^{(d)}_{m,n}(e^{(d)}, u, v) = \mathrm{MAP}(P),$$

where

$$P = \prod_{1\le i,j\le n} \prod_{\delta=1}^{d} (1 - u_i e^{(\delta)}_{i,j}) \prod_{1\le i,j\le n} \prod_{\delta=1}^{d} (1 - e^{(\delta)}_{i,j} v_j)$$

is the product from Lemma 6.4.

Proof. Follows immediately from Lemma 6.4: After applying the MAP-operator to the righthand side, only the multilinear monomials survive. The multilinear monomials are exactly the monomials of $DFext^{(d)}_{m,n}$. The sign of a monomial in P is positive if it contains an even number of edge variables $e^{(\delta)}_{i,j}$. Otherwise it is negative. The same is true in $DFext^{(d)}_{m,n}$. Therefore, also the signs match. □

Corollary 6.6. $DFext^{(d)}_{m,n}$ *is real stable.*

Proof. Follows from the facts that the polynomial P is real stable by Lemma 6.2 and that the operator MAP is stability preserving. □

From $DFext^{(d)}_{m,n}$, we can obtain $DF^{(d)}_{m,n}$ by setting the u and v variables to 1, that is $DF^{(d)}_{m,n}(e^{(d)}) = DFext^{(d)}_{m,n}(e^{(d)}, u, v)|_{u_1,\dots,u_n,v_1,\dots,v_n=1}$.

Lemma 6.7. $DF^{(d)}_{m,n}$ *is real stable.*

Proof. Replacing variables by positive real numbers preserves stability, see e.g. (Pemantle, 2012). □

Since monomials of $DF^{(d)}_{m,n}$ stand in one-to-one correspondence with double functions in $K^{(d)}_{m,n}$, we can count all double functions (with signs) in a given multigraph G by setting the edge variables $e^{(\delta)}_{i,j}$ to 1 or 0, depending whether the edges appear in G or not.

7. Complexity theory basics

We give some background information on the relevant complexity classes and results. We refer to (Papadimitriou, 1994; Arora & Barak, 2009) for further explanations and proofs of the well-known theorems mentionend in this section. Understanding the exact definitions of the classes is not necessary to understand the paper, the important fact for this paper is that counting perfect matchings in bipartite graphs is hard.

Deciding whether a formula ϕ in conjunctive normal form (CNF) has a satisfying assignment is the defining problem of the famous class NP. If we instead want to count the number of satisfying assignments, we get a problem which is complete for the class #P defined by Valiant. Obviously, when you can count the number of satisfying assignments, then you can decide whether there is at least one, therefore, #P is a "harder" class than NP. It turns out that some problems become #P-hard when considered as a counting problem while their decision versions are easy. Perfect matchings in bipartite graphs is such an example: There are efficient algorithms for the decision problem, but the counting version is hard, as proven by Valiant.

Theorem 7.1 (Valiant)**.** *Counting perfect matchings in bipartite graphs is* #P*-complete under Turing reductions.*

Above, a problem A is Turing reducible to a problem B if there is a polynomial time deterministic Turing machine that solves A having oracle access to B. This means, that an efficient algorithm for B will yield an efficient algorithm for A and in this sense, A is easier than B.

The class NP is defined by an existential quantifier: There is an assignment that satisfies the given formula. The complement coNP of NP is defined by a universal quantifier: Every assignment is not satisfying. If we now take more quantifiers, we get the so-called polynomial time hierarchy PH, an ascending chain of complexity classes Σ_k (and the corresponding co-classes Π_k). It is widely believed that PH is infinite, that is, $\Sigma_k \subsetneq \Sigma_{k+1}$ for all k. If for some k, $\Sigma_k = \Sigma_{k+1}$, then we say that PH collapses, since in this case it can be shown that $\Sigma_k = \mathsf{PH}$. Experts consider this to be very unlikely. Therefore, if some assumption implies that PH collapses, this is a strong indication that this assumption is false.

The class P/poly is a so-called nonuniform class. A problem P is in the class P/poly if there is a polynomial time deterministic Turing machine M and a sequence of polynomially long strings (a_n), the so-called "advice", such that for all inputs x of length n, M on input (x, a_n) accepts iff $x \in P$. That is, in addition to the input, we get the advice a_n, which has to be the same for all inputs of length n. Alternatively, one can define P/poly as the set of all problems that have polynomial size nonuniform circuits.

Theorem 7.2 (Karp–Lipton)**.** *If* NP $\subseteq$ P/poly*, then* PH *collapses to the second level* Σ_2*.*

8. Counting double functions is #P-hard

We prove that counting double functions is #P-hard by reducing the problem of counting perfect matchings PM in bipartite graphs to it. We start by showing that the counting of *total* double functions can be reduced to the counting of all double functions.

Lemma 8.1. *The evaluation of $DFtot^{(d)}_{m,n}$ is polynomial time reducible to the evaluation of $DF^{(d)}_{m,n}$ (by Turing reductions).*

Proof. Every monomial of $DFtot^{(d)}_{m,n}$ also appears in $DF^{(d)}_{m,n}$. Let M be a monomial in $DF^{(d)}_{m,n}$ corresponding to two functions F and H. The degree of M is $|\operatorname{dom} F| + |\operatorname{dom} H|$, the sum of the size of the domains of the two functions. If M corresponds to two total functions (and hence appears in $DFtot^{(d)}_{m,n}$), then the total degree of M is $m + n$. All other monomials have smaller total degree.

We replace each variable $e^{(\delta)}_{i,j}$ by $e^{(\delta)}_{i,j} \cdot t$, where t is a new variable for bookkeeping purposes. We consider the resulting polynomial $\hat{DF}^{(d)}_{m,n}$ as a polynomial in t with coefficients being polynomials in the original variables $e^{(\delta)}_{i,j}$. Monomials of the same total degree s in $DF^{(d)}_{m,n}$ will all appear in the coefficient polynomials of t^s.

Therefore, we plug in $m + n + 1$ many different values for t into $\hat{DF}^{(d)}_{m,n}$. From these values we can obtain the coefficients of $\hat{D}F^{(d)}_{m,n}$ (as a polynomial in t) using Lagrange interpolation with only polynomial overhead. The coefficients of t^{m+n} is $DFtot^{(d)}_{m,n}$. □

Remark 8.2. In the proof above, we can assume that we are given a PGC for $DF^{(d)}_{m,n}$ and from this, we construct a PGC for $DFtot^{(d)}_{m,n}$ whose size is polynomial in the size of the given circuit, since we need $m + n + 1$ copies of the circuit for the evaluation at the interpolating points and the circuit that realizes the Lagrange interpolation. The same proof would also work if we only could evaluate $DF^{(d)}_{m,n}$ by oracle access. When we want to compute a particular value $DFtot^{(d)}_{m,n}(e)$ for some input point e, then we query $DF^{(d)}_{m,n}(\tau \cdot e)$ for $m + n + 1$ different scalar values of τ and can recover $DFtot^{(d)}_{m,n}(e)$ using the same interpolation process.

Theorem 8.3. *Counting of perfect matchings in bipartite graphs is polynomial time reducible to the evaluation of $DFtot^{(n+2)}_{n,n}$. The evaluation points occuring in the reduction are all $\{0, 1\}$-valued.*

The proof is deferred to Appendix A due to space constraints.

Remark 8.4. Again, the reduction works when we are given a PGC for $DFtot^{(n+2)}_{n,n}$. From this, we can compute a PGC for PM in polynomial time. Or $DFtot^{(n+2)}_{n,n}$ is given as an oracle, then we get a Turing reduction from counting perfect matchings to $DFtot^{(n+2)}_{n,n}$.

Corollary 8.5. *Evaluating $DFtot^{(n+2)}_{n,n}$ is $\#$P-hard. This is even true when the input is $\{0, 1\}$-valued.*

Proof. This follows from Theorem 8.3 together with the fact that counting perfect matchings is $\#$P-hard. □

9. Normalization

While we now have a real stable polynomial that is $\#$P-hard to evaluate, namely $DF^{(n+2)}_{n,n}$, by Lemma 6.7, Lemma 8.1 and Corollary 8.5, $DF^{(n+2)}_{n,n}$ is not the generating polynomial of a SR distribution. First of all, some of the coefficients are negative and second, the distribution is not normalized.

Note that the signs of the monomials of $DF^{(d)}_{m,n}$ have a regular pattern, the signs of odd degree monomials are negative and of even degree monomials are positive. For a polynomial $f(x_1, \dots, x_n)$ with the degree of x_1 being t in f, the inversion of f at x_1 is $x_1^t f(-1/x_1, x_2, \dots, x_n)$. Note that this is again a polynomial and the degree of x_1 is $\le t$. The inversion of f is the polynomial obtained when inverting every variable.

Lemma 9.1. *Let P_n be the inversion of $DF^{(n+2)}_{n,n}$. Then*

1. *P_n is multilinear.*
2. *All cofficients of P_n are nonnegative.*
3. *P_n is real stable.*

Proof. The first item follows from the fact, that the inversion cannot increase the degree of a variable. For the second item note that the inversion flips the sign of each variable. Since the coefficients of the odd degree monomials have negative signs in $DF^{(d)}_{m,n}$, they have positive signs in P_n. The even degree monomials have positive coefficients and they stay positive in P_n. The third item follows, since inversion preserves stability, see e.g. (Pemantle, 2012). □

So the coefficients of P_n are an unnormalized SR distribution.

Lemma 9.2. *The evaluation of P_n is $\#$P-hard, even when restricted to integers of size polynomial in n.*

Proof. $DFtot^{(n+2)}_{n,n}$ is $\#$P-hard to evaluate, even when restricted to $\{0, 1\}$-valued vectors by Theorem 8.3. In the reduction of $DFtot^{(n+2)}_{n,n}$ to $DF^{(n+2)}_{n,n}$ we are free to choose the interpolation points, so we could choose values of the form $1/s$. In the reduction of $DF^{(n+2)}_{n,n}$ to P_n, $1/s$ will be replaced by $-s$, so the claim follows. □

$S_n := P_n(1, \dots, 1)$ is the sum of all coefficients of P_n. S_n can be computed efficiently, since it is the number of all double functions of $K^{(n+2)}_{n,n}$, see the next lemma. Let $\hat{P}_n := P_n/S_n$. $\hat{P}_n$ now is a normalized SR distribution that has all the properties from Lemmas 9.1 and 9.2.

We now show how to compute S_n in polynomial time. We need to introduce an auxiliary quantity. $T_{m,n,\ell}$ denotes the number of all instantiations of double functions (F,H) in $K^{(d)}_{m,n}$ such that $\operatorname{dom} F \subseteq \{1,\dots,\ell\}$. Consider such a double function. We can have $F(s) = n$ for any number $j \le \ell$ of indices $s \in \{1,\dots,\ell\}$. For each s, we can choose one out of d multiedges. On the other hand, $H(n)$ can either be undefined, or mapped to one of the j indices s with $F(s) = n$ or mapped to another index. In the first case, there is one choice, in the second case we can choose out of $d-1$ multiedges and in the third out of d. This gives the following recursion

$$T_{m,n,\ell} = \sum_{j=0}^{\ell} \binom{\ell}{j} d^j (1 + j(d-1) + (m-j)d) T_{m,n-1,\ell-j}$$

for $n > 0$ and $\ell > 0$. The status of the node v_n is completely determined, so we can remove it from the further considerations. The domain of the residual function F' is—after reordering—contained in $\{1,\dots,\ell-j\}$. For the base cases, we have

$$T_{m,0,\ell} = 1,$$
$$T_{m,n,0} = (md+1)^n.$$

In the case when $n = 0$, then there is only one double function (F,H). F is undefined everywhere and H is the empty function. In the case when $\ell = 0$, then there is only one F, again the function that is undefined everywhere. But H can be an arbitrary function, for each $s \in \{1,\dots,n\}$, we can either choose one out of the md multiedges leaving v_s or have $H(s)$ being undefined.

Lemma 9.3. *We can compute the normalization constant S_n in polynomial time.*

Proof. We have $S_n = T_{n,n,n}$. The entries in the table $T_{m,\nu,\lambda}$ can be computed using two nested loops running over ν and λ . The computation of one value $T_{m,\nu,\lambda}$ can be done in polynomial time. □

Note that for the further proof it would not be necessary that we can compute S_n, since we could also encode this knowledge in the advice string.

10. Circuit lower bounds

Recall that we want to prove that there are SR distributions that have large (that is superpolynomial) PGCs. Recall that $\hat{P}_n$ is the generating polynomial of an SR distribution.

Theorem 10.1. *If $\hat{P}_n$ has PGCs of polynomial description size, then* $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{P/poly}$.

Proof. Let M be a polynomial time deterministic Turing machine with oracle access to a problem in $\#\mathsf{P}$. We construct a polynomial time DTM with polynomial advice that can simulate M. The advice a_n will be the PGC for P_n. N simulates M step by step. Whenever M calls the oracle, M will evaluate the PGC a_n instead. This is possible, since evaluating $\hat{P}_n$ is $\#\mathsf{P}$-hard. If the PGC has high degree, then the intermediate results might become large. In this case, we multiply by S_n (see Lemma 9.3), get the integer valued polynomial P_n and can now perform the computation modulo an integer with polynomially many bits. (This is a standard technique.) The result will be exact, since $P_n(e)$ has polynomially many bits. □

We finally obtain our main result, that there are SR distributions without polynomial size PGCs, unless PH collapses.

Corollary 10.2. *If $\hat{P}_n$ has PGCs of polynomial description size, then* PH *collapses.*

Proof. By Toda's theorem (see Section 17.4 of (Arora & Barak, 2009)), $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$. Since $\mathsf{NP} \subseteq \mathsf{PH}$, the corollary follows directly from the Karp–Lipton theorem (Theorem 7.2). □

References

Amini, N. Stable multivariate generalizations of matching polynomials, 2019. URL https://arxiv.org/abs/1905.02264.

Arora, S. and Barak, B. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

Borcea, J. and Brändén, P. The Lee-Yang and Pólya-Schur programs. I. Linear operators preserving stability. *Inventiones mathematicae*, 177:541–569, 2009.

Borcea, J., Brändén, P., and Liggett, T. M. Negative dependence and the geometry of polynomials. *J. Amer. Math. Soc.*, 22:521–567, 2009.

Borodin, A. and Rains, E. M. Eynard–Mehta theorem, Schur process, and their Pfaffian analogs. *Journal of Statistical Physics*, 121:291–317, 2005.

Bouchet, A. and Cunningham, W. H. Delta-matroids, jump systems, and bisubmodular polyhedra. *SIAM Journal on Discrete Mathematics*, 8(1):17–32, 1995. doi: 10.1137/S0895480191222926. URL https://doi.org/10.1137/S0895480191222926.

Choe, Y., Oxley, J., Sokal, A., and Wagner, D. Homogeneous multivariate polynomials with the half-plane property. *Advances in Applied Mathematics*, 32(1-2):88–187, 2004. ISSN 0196-8858. doi: 10.1016/S0196-8858(03)00078-2.

Darwiche, A. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

Heilmann, O. J. and Lieb, E. H. Theory of monomer-dimer systems. *Communications in Mathematical Physics*, 25 (3):190—-232, 1972.

Kisa, D., Van den Broeck, G., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. In Baral, C., Giacomo, G. D., and Eiter, T. (eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014. URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8005.

Koller, D. and Friedman, N. *Probabilistic Graphical Models Principles and Techniques*. MIT Press, 2009.

Kulesza, A. and Taskar, B. Determinantal point processes for machine learning. *Found. Trends Mach. Learn.*, 5 (2-3):123–286, 2012. doi: 10.1561/2200000044. URL https://doi.org/10.1561/2200000044.

Martens, J. and Medabalimi, V. On the expressive efficiency of sum product networks. *CoRR*, abs/1411.7717, 2014. URL http://arxiv.org/abs/1411.7717.

Meila, M. and Jordan, M. I. Learning with mixtures of trees. *J. Mach. Learn. Res.*, 1:1–48, 2000. URL http://jmlr.org/papers/v1/meila00a.html.

Papadimitriou, C. *Computational Complexity*. Addison Welsey, 1994.

Pemantle, R. Hyperbolicity and stable polynomials in combinatorics and probability. In *Current Development in Mathematics, Proceedings of the 2011 conference*, pp. 57–124. International Press, 2012.

Poon, H. and Domingos, P. M. Sum-product networks: A new deep architecture. *CoRR*, abs/1202.3732, 2012. URL http://arxiv.org/abs/1202.3732.

Roth, D. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(94)00092-1. URL https://www.sciencedirect.com/science/article/pii/0004370294000921.

Saptharishi, R. et al. A selection of lower bounds in arithmetic circuit complexity, 2021. URL https://github.com/dasarpmar/lowerbounds-survey/releases/tag/v9.0.3. Version 2021-07-27.

Zhang, H., Juba, B., and Van den Broeck, G. Probabilistic generating circuits. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12447–12457. PMLR, 2021. URL http://proceedings.mlr.press/v139/zhang21i.html.

A. Omitted proofs

Proof. (of Theorem 8.3) Let $G = (U \cup V, E)$ be an instance of PM. G has bipartition U and V. We can assume that $|U| = |V| = n$. We label the nodes in both sets by $\{1, \dots, n\}$.

We consider $DFtot^{(d)}_{n,n}$ on $K^{(d)}_{n,n}$. (The value d will be chosen later and it turns out that $n+2$ is the right choice.) We encode the input graph G into $K^{(d)}_{n,n}$ by setting all variables $e^{(\delta)}_{i,j}$ to zero, $1 \le \delta \le d$, for all edges $(i,j) \notin E$. That is, we get a multigraph $\hat{G}$ in which each edge of G is duplicated d times.

Now take a total double function (F, H) of G. We have $n = |F| = |H|$, so there are $2n$ edges in total. How many ways are there to map it into $\hat{G}$? Assume that F and G share s edges. If an edge appears only in F or only in G, then we can choose one of the d copies in $\hat{G}$. There are $2n - 2s$ edges of this kind. If an edge appears both in F and H, then for first edge, we can choose one of the d copies, but for the second edge, we can only choose from the remaining $d-1$ copies, since by the definition of $DFtot^{(d)}_{n,n}$, we have to select distinct edges. Thus is F and H share s edges, then the pair (F, H) gives rise to

$$d^{2n-2s}(d(d-1))^s$$

instantiations in $\hat{G}$.

What pairs (F, H) share the most edges? These are the pairs with $F = H$. This means that F is a function from $U \to V$ but also from $V \to U$. Since $|U| = |V|$ and F is total, this can only happen if F is a perfect matching in G, that is, a bijective function. Perfect matchings and double functions sharing n edges stand in a one-to-one correspondence, since for each perfect matching M, (M, M) is obviously a total double function.

Let a_s be the number of double functions in G that share s edges. Let $D^{(d)}_{\hat{G}}$ be the evaluation of $DFtot^{(d)}_{n,n}$ at $\hat{G}$, that is, we plug in the value 0 for each multiedge that is not present in $\hat{G}$ and the value 1 for each multiedge that is present in $\hat{G}$. By the consideration above, we have

$$D^{(d)}_{\hat{G}} = \sum_{s=0}^{n} d^{2n-2s}(d(d-1))^s a_s = d^n \sum_{s=0}^{n} d^{n-s}(d-1)^s a_s. \tag{6}$$

The last equations shows that $DFtot^{(d)}_{n,n}$ viewed as a function in d is a polynomial in d. Note that we can get any value $D^{(t)}_{\hat{G}}$ from $DFtot^{(d)}_{n,n}$ for any value $t \le d$ by setting the variables $e^{(1)}_{i,j} = \cdots = e^{(t)}_{i,j} = 1$ and $e^{(t+1)}_{i,j} = \cdots = e^{(d)}_{i,j} = 0$ for all edges (i,j) in $\hat{G}$ and all edge variable for edges not in $\hat{G}$ to 0.

Since d is arbitrary, Equation (6) is also true other values. We can write this in matrix form:

$$\begin{pmatrix} D^{(2)}_{\hat{G}} \\ D^{(3)}_{\hat{G}} \\ \vdots \\ D^{(n+2)}_{\hat{G}} \end{pmatrix} = \begin{pmatrix} 2^n & 0 & \dots & 0 \\ 0 & 3^n & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & (n+2)^n \end{pmatrix} \begin{pmatrix} 2^n & 2^{n-1} & \dots & 1 \\ 3^n & 3^{n-1}2 & \dots & 2^n \\ \vdots & \vdots & \ddots & \vdots \\ (n+2)^n & (n+2)^{n+1}(n+1) & \dots & (n+1)^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$$

$$= \begin{pmatrix} 2^n & 0 & \dots & 0 \\ 0 & 3^n 2^n & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & (n+2)^n(n+1)^n \end{pmatrix} \begin{pmatrix} 2^n & 2^{n-1} & \dots & 1 \\ (\frac{3}{2})^n & (\frac{3}{2})^{n-1} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ (\frac{n+2}{n+1})^n & (\frac{n+2}{n+1})^{n+1} & \dots & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$$

The matrix on the lefthand side in the second row is a diagonal matrix with nonzero elements on the diagonal, hence it is invertible. The matrix on the righthand side is a Vandermonde matrix. Since the values $2, \frac{3}{2}, \dots, \frac{n+2}{n+1}$ are all pairwise distinct, the second matrix is also invertible. Hence we can compute the values $a_0, \dots, a_n$ from $D^{(2)}_{\hat{G}}, D^{(3)}_{\hat{G}}, \dots, D^{(n+2)}_{\hat{G}}$ in polynomial time.

Thus, the overall reduction works as follows: Given our input graph G, we compute the values $D^{(2)}_{\hat{G}}, D^{(3)}_{\hat{G}}, \dots, D^{(n+2)}_{\hat{G}}$ by specialising the variables $e^{(\delta)}_{i,j}$ as described above. From these values, we can recover a_n, which is the number of perfect matchings in G. $\square$