Computation of Determinant vs. Permanent

A Survey

Hadley Black

April 2015

1 Introduction

A major topic in computational complexity theory is the Determinant vs. Permanent problem. Given an $n \times n$ matrix $A$, the determinant and permanent are polynomials defined on the entries of $A$:

\[
\text{det } A = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}
\]

\[
\text{perm } A = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma_i}.
\]

$\sigma \in S_n$ denotes a permutation (or reordering function) of $\{1, 2, ..., n\}$ and $\text{sgn}(\sigma)$ is the “signature” value of $\sigma$. $\text{sgn}(\sigma) = 1$ if $\sigma$ can be achieved by an even number of pairwise element interchanges and $\sigma = -1$ otherwise.

The first thing to notice is that, disregarding the sign of each term, the determinant and permanent are the exact same polynomial. However, despite their surface level resemblance, they are strongly different in their inherent complexity. While the determinant is computable in polynomial time, the permanent has been shown to be $\#P$-hard.

The Determinant vs. Permanent problem has been of particular interest as it gives an example of an “easy” problem, which under some seemingly insignificant alteration becomes vastly more difficult to solve. In sections 2 and 3 I will discuss and prove theorems regarding the complexity of each problem. In section 4 I explore some important combinatorial interpretations of determinant and permanent to point out one of many connections this problem has to other areas of mathematics.

2 Complexity of the Determinant

At first it is not obvious why computing the determinant should be possible in polynomial time. By definition it is a sum over all possible products taken by, for each column, choosing a term from a unique row. The result is an $n!$ term summation. The key insight that lets us escape this fact and leads to an efficient computation is the determinant’s homomorphic
property, i.e. \( \det AB = \det A \det B \). Now, if we could find a collection of simpler matrices \( \{A_i\}_{i=1}^k \) for which \( A = \prod_{i=1}^k A_i \), where \( \det A_i \) is “easy” to compute and \( k \) is a number which scales polynomially with the size of \( A \), then \( \det A = \prod_{i=1}^k \det A_i \) and this may give us a polynomial time algorithm.

2.1 A Polynomial Algorithm

**Theorem 2.1.** The determinant can be computed in polynomial time.

**Proof.** To decompose \( A \) into a collection \( \{A_i\}_{i=1}^k \) for which \( A = \prod_{i=1}^k A_i \), we make use of Gaussian elimination. We will use a series of row operations to transform \( A \) into an upper triangular matrix (where each term below the diagonal is zero). Each of these row ops will correspond to multiplying \( A \) by some matrix \( B \) whose determinant is easy to compute. Finally, the determinant of an upper triangular matrix is simply the product of the diagonal entries, since any other permutation necessarily will include one of the zeros below the diagonal.

To transform \( A \) into an upper triangular matrix we follow an \( n \)-step algorithm where at step \( j \) we seek to put a non-zero value at \( A_{jj} \) and set \( A_{ij} = 0, \forall i > j \). On step \( j \) suppose that entries below the diagonal and prior to column \( j \) are zero. Now do the following:

1. If \( A_{jj} = 0 \), do a row swap with some row \( i > j \) for which \( A_{ij} \neq 0 \). If this is not possible and we must set \( A_{jj} = 0 \), then it follows that there will be a zero entry along the diagonal and the determinant will necessarily be zero.

2. Now, for each row \( A_i, i > j \) for which \( A_{ij} \neq 0 \) do the following row addition: \( A_i \leftarrow A_i - A_j \cdot (A_{ij}/A_{jj}) \). Thus, the entry \( A_{ij} \) is set to zero as desired.

Our algorithm works in \( n \) steps where the \( j \)th step requires at most \( n - j \) row ops, each of which is a calculation of \( n \) terms. Thus, transforming \( A \) into an upper triangular matrix can be done in

\[
n \cdot \left( n + (n - 1) + (n - 2) + \cdots + 2 + 1 \right) = O(n^3).
\]

Now let the \( i \)th row op correspond to multiplying \( A \) by a matrix \( B_i \). Denote \( A \) in its upper triangular form as \( A' \) and let \( k \) be the number of row ops. Then,

\[
A' = A \cdot \prod_{i=1}^k B_i
\]

and so

\[
\det A' = \det A \cdot \prod_{i=1}^k \det B_i \tag{1}
\]

If \( B_i \) is a row swap operation, we know \( \det B_i = -1 \) and for the addition operations \( \det B_i = 1 \). Thus, we can rewrite (1) as
\[ \det A = \det A' \cdot \prod_{i=1}^{k} \det B_i. \]  

Finally since \( \det A' \) is simply the product of its diagonal entries and we know \( \det B_i, \forall i \), \( \det A \) is easily retrieved from (2). Thus, using this algorithm we can compute the determinant of a matrix in \( O(n^3) \) and so this problem is in \( \mathbb{P} \).

\[ \square \]

3 Complexity of the Permanent

In this section I will first provide and discuss some basic definitions regarding the \( \#\mathbb{P} \) complexity class. I will then give a proof that the general permanent is \( \#\mathbb{P} \)-hard. This will lead to a proof of Valiant’s Theorem, a seminal result in complexity theory from Leslie Valiant in 1979.

3.1 Complexity of Counting and \( \#\mathbb{P} \)

We define \( \mathbb{NP} \) as the set of problems for which solutions can be verified in polynomial time. A rigorous definition is as follows:

**Definition 3.1.** \( \mathbb{NP} \) is the class of functions \( A(x) \) of the form

\[ A(x) = \exists c : V(x,c) \]

where \( V \in \mathbb{P} \) and \( c = \text{poly}(|x|) \).

By this definition, \( A \in \mathbb{NP} \) means that we have an algorithm \( V \) which decides if a certificate (possible solution) \( c \) satisfies a problem instance \( x \) in poly-time. A problem in \( \mathbb{NP} \) asks if an object with a certain property exists and in turn has a poly-time method for checking if a given object \( c \) indeed has that property. For each problem of this form one could imagine the corresponding problem of instead counting the objects having the property. Presumably, a problem of this form should be much harder since finding any single solution answers the \( \mathbb{NP} \) variant. We call this new class \( \#\mathbb{P} \) and define it as follows:

**Definition 3.2.** \( \#\mathbb{P} \) is the class of functions \( A(x) \) where

\[ A(x) = |\{ c : V(x,c) \}| \]

such that \( V \in \mathbb{P} \) and \( |c| = \text{poly}(|x|) \).

While an \( \mathbb{NP} \) problem can be regarded as a search for a needle in a haystack, a corresponding \( \#\mathbb{P} \) problem demands to know how many needles there are.

Now, since \( \#\mathbb{P} \) is not a class of decision problems it is necessary to define a slightly different notion of reduction:
**Definition 3.3.** If $A, B$ are problems in $\#P$, a counting reduction from $A$ to $B$ is a pair of functions $f, g \in \mathbb{P}$ such that

$$\forall x, A(x) = g(x, B(f(x))).$$

If this is the case, then $A \leq_{poly} B$, i.e. $B$ is at least as hard as $A$.

Now $A \leq_{poly} B$ means that we can run $B$ on a transformed problem instance of $A$ and then transform the answer $B$ outputs to get the correct solution for $A$. Note that these transformations must be poly-time.

### 3.2 Permanent is $\#P$-hard

To prove that computing the permanent is $\#P$-hard we will show that it must be at least as hard as the counting version of the well known $\mathbb{NP}$-complete problem, 3-SAT. We will call this problem $\#3$-SAT and it will ask how many possible satisfying variable assignments there are for a CNF formula $\phi$ with $n$ variables and $m$ clauses (and 3 literals per clause).

We begin by showing that the permanent computes the sum of the weights of all cycle covers in a weighted digraph. With this in mind and given an instance of $\#3$-SAT we construct a weighted digraph whose permanent will be a function of $\#\phi$ (the number of satisfying assignments of $\phi$). We define two types of “gadgets” which are joined together to form the larger graph. Each variable will have a corresponding vertex which is on exactly two cyclic paths composed of edges of weight 1. Each cycle cover will contain exactly one of these cycles, which correspond to setting the truth value of the variable.

**Theorem 3.1.** Computing the permanent is $\#P$-hard.

**Claim i)** Given the adjacency matrix $A$ for a weighted $n$-vertex digraph $G$ (with possible self-loops), $\text{perm}(A)$ is the sum of the weights of all cycle covers of $G$.

**Proof (see [2, p. 157]).** Let $A$ be the $n \times n$ adjacency matrix for a weighted digraph $G$. First, notice that $A_{ij} = 1$ means by definition that there is a corresponding edge in $G$, i.e. $(i,j) \in E_G$. Now, let $\sigma$ be some permutation on $1, 2, ..., n$ where $\sigma_i$ gives that $i$th element of $\sigma$. Observe that $\prod_{i=1}^{n} A_{i,\sigma_i} \neq 0$ if and only if $(1, \sigma_1), (2, \sigma_2), ..., (n, \sigma_n) \in E_G$. In each sequence $(i)_{i=1}^{n}$ and $(\sigma_i)_{i=1}^{n}$, each vertex appears exactly once. Thus, the subgraph induced on edges $(i, \sigma_i)$ has the property that every vertex is incident to exactly two edges. More formally, $\forall i \in V_G, \deg(i) = 2$. This is precisely the definition of a cycle cover. We define the weight of a cycle cover to be product of its edge weights.

Thus, if $\sigma$ induces a valid subgraph of $G$, that subgraph is necessarily a cycle cover with weight $\prod_{i=1}^{n} A_{i,\sigma_i} \neq 0$. If $\sigma$ does not induce a valid subgraph of $G$ (i.e. $\exists (i, \sigma_i) \notin E_G$) then $\prod_{i=1}^{n} A_{i,\sigma_i} = 0$. So $\text{perm}(A)$ is the sum of the weights of all cycle covers of $G$.\[\square\]
Claim ii) Given a boolean formula \( \phi \) it is possible to construct a graph \( G \) whose adjacency matrix \( A \) satisfies \( \#\phi = f(\text{perm}(A)) \) where \( f \) is some poly-time function and \( |A| = \text{poly}(|\phi|) \).

Proof (see [1, p. 664] [2, p. 158]). Constructing a graph from \( \phi \):

1. Variable gadget: For each variable, construct a gadget with two cyclic paths passing through it. One path corresponds with a “true” assignment while the other corresponds with a “false” assignment. Each variable gadget must satisfy the following:

   (a) A cycle cover must include \textit{exactly one} of these paths so that truth assignments are fully consistent.

   (b) We then “thread” the paths associated with a variable \( x \) through each clause gadget containing the literal \( x \) or \( \overline{x} \).

2. Clause gadget: Let each clause gadget be a graph on 3 “input” vertices as well as some other internals which will be described later. Each input vertex corresponds to a unique literal in the clause.

   (a) If the literal \( x \) or \( \overline{x} \) appears in a clause, let the corresponding input vertex be on the cyclic path associated with “true” or “false” for \( x \), respectively.

   (b) We must design the clause gadget such that it contributes a weight of some constant \( C \) whenever \textit{at least one} of its “input” vertices is covered (i.e. if the clause is satisfied). Otherwise, we need the total weight to be 0.

   (c) Additionally, we require that the total weight is 0 \textit{if the following does not hold}: For each input vertex, its incoming edge is covered \textit{if and only if} its outgoing edge is covered. That is, each variable path is either entirely covered, or entirely uncovered.

Consider some truth assignment \( \sigma \) on the \( n \) variables of \( \phi \). In each clause gadget input vertices associated with true assigned literals are covered, but to achieve a full cycle cover we need also to cover the other input variables as well as the internals of the clause gadgets. The total contribution of \( \sigma \) to the permanent will then be the product, over all clauses, of the total weight of their internal cycle covers.

If we can design a clause gadget satisfying the outlined conditions, then each satisfying assignment of \( \phi \) will contribute \( C^m \) to the permanent where \( m \) is the number of clauses. If an assignment does not satisfy \( \phi \), then some clause will have an internal cycle cover weight of 0, making the full product over each clause evaluate to 0.

Now, let \( M \) be the adjacency matrix for the clause gadget. The first three vertices (rows and columns) will correspond to the input vertices, while the rest will represent the internal vertices. Suppose \( M \) is \( k \times k \) and \( S, T \subseteq \{1, 2, \ldots, k\} \). We define \( M^{(S,T)} \) to be the matrix resulting from removal of the \( i \)th row for all \( i \in S \) and the \( j \)th column for all \( j \in T \). Now, let \( S, T \subseteq \{1,2,3\} \). \( S \) represents some collection of incoming edges to the input vertices while \( T \) represents some collection of outgoing edges from the input vertices. Thus, if \( S \neq T \), then the clause consistency condition (2c) has been violated. If \( S = T = \emptyset \) then the clause is unsatisfied by the given assignment and thus \( \phi \) is unsatisfied. Finally, if \( S = T \neq \emptyset \) this
indicates that the clause is satisfied. With this in mind we want to find a matrix $M$ which satisfies

$$\forall S, T \subseteq \{1, 2, 3\}, \text{perm}(M^{(S,T)}) = \begin{cases} 0 & \text{if } S = T = \emptyset \\ C & \text{if } S = T \neq \emptyset \\ 0 & \text{if } S \neq T \end{cases}.$$ 

It can be verified that

$$M = \begin{pmatrix} 0 & 0 & -1 & -1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 2 & -1 & 1 & 1 \\ 0 & 0 & 0 & -1 & -1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 2 & -1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

satisfies the desired properties with $C = 12$. Thus, we let $M$ be the adjacency matrix which describes the clause gadget.

Now, every cycle cover in $G$ corresponds to some assignment of truth values to the $n$ variables. Each cycle cover corresponding to an unsatisfying or inconsistent assignment has weight 0. On the other hand, the sum of the weights of valid satisfying assignments of each clause is 12. Since there are $m$ clauses, a satisfying assignment then corresponds to a cycle cover of weight $12^m$. Thus the total sum of the weights of all cycle covers in $G$ is $12^m \# \phi$. Thus, $\text{perm}(A) = 12^m \# \phi$ and so

$$\#3\text{-SAT} \leq_{poly} \text{PERMANENT}.$$ 

### 3.3 Valiant’s Theorem

Now that we have proven that the permanent is $\#P$-hard it is an interesting result that this problem retains its complexity on certain types of seemingly simpler matrices. In 1979, Leslie Valiant published his proof that computing the permanent remains $\#P$-hard on matrices whose entries are all 0 or 1. It turns out that the problem of counting the number of perfect matchings in a bipartite graph is reducible to computing the permanent of a 0-1 matrix. On the other hand, the corresponding search problem of finding a single perfect matching in a bipartite graph can be done in polynomial time. Thus Valiant’s theorem gives an example of an “easy” problem that becomes vastly more complex when one attempts to count solutions.

**Theorem 3.2.** [Valiant’s Theorem] Computing the permanent of a 0-1 matrix is $\#P$-hard.

We can prove this result in two steps. We begin by reducing PERMANENT to the variant POS-PERMANENT for a matrix of non-negative entries.
Claim i) \( \text{PERMANENT} \leq_{\text{poly}} \text{POS-PERMANENT} \).

Proof (see [1, p. 667]). Let \( A \) be a matrix with \( n \)-bit integer entries. Since the \( \text{perm}(A) \) is an \( n! \) term summation over \( n \)-length products of entries in \( A \), we can derive the following naive bound:

\[
|\text{perm}(A)| \leq n! \left( \max_{i,j} |A_{ij}| \right)^n \leq n! \cdot 2^{n^2} < (2^{n^2})^2 = 2^{2n^2}.
\]

Denote this quantity as \( Q \) and define a new matrix \( A' \) as \( A'_{ij} = A_{ij} \mod 2Q \). \( A' \) is a matrix of non-negative entries where:

\[
A'_{ij} = \begin{cases} 
A_{ij} & \text{if } A_{ij} \geq 0 \\
2Q + A_{ij} & \text{if } A_{ij} < 0.
\end{cases}
\]

Note that \( Q \) has \( \mathcal{O}(n^2) \) bits and so \( A' \) is an instance of POS-PERMANENT of size \( \text{poly}(|A|) \).

Moreover, \( \text{perm}(A) \mod 2Q = \text{perm}(A') \mod 2Q \). Call this quantity \( R \). Now, since \( |\text{perm}(A)| < Q \), we have

\[
\text{perm}(A) = \begin{cases} 
R & \text{if } R < Q \\
R - 2Q & \text{if } R \geq Q.
\end{cases}
\] (3)

Thus, \( \text{PERMANENT} \leq_{\text{poly}} \text{POS-PERMANENT} \).

In summary, given \( A \) with possibly negative entries we can construct \( A' = f(A) \) such that \( A' \) has all positive entries and where \( f \) is a poly-time transformation. We can then compute \( \text{perm}(A') \) and “decode” the result by use of a poly-time function \( g \) as described by eq. (7) to get \( \text{perm}(A) \). That is, \( \text{perm}(A) = g(\text{perm}(A')) \). Thus, POS-PERMANENT is \#P-hard.

Our next step is to reduce 0-1 PERMANENT to POS-PERMANENT. To do this we let \( A \) be a matrix of positive entries and consider the graph \( G \) with \( A \) as its adjacency matrix. We then simulate the edges in \( G \) using gadgets composed of weight 1 to construct a new graph \( G' \). Our goal is to build \( G' \) so that the sum of the weights of its cycle covers is the same as that of \( G \) and thus the permanents of their respective adjacency matrices are equal.

Claim ii) \( \text{POS-PERMANENT} \leq_{\text{poly}} 0\text{-1 PERMANENT} \).

Proof (see [1, p. 667] [2, p. 160]). Let \( G \) be the graph with \( A \) as its adjacency matrix where the entries of \( A \) are non-negative. Consider some edge \( e = (u, v) \) in \( G \) and suppose it has weight \( w \). For each \( 2^k \) term in the binary representation of \( w \) create a chain of \( k \) links where each link has 2 cycle covers. More precisely, we know \( \exists k_1, k_2, \ldots, k_t \) such that \( w = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_t} \). Let \( G' \) have the following gadget in place of the edge \( e = (u, v) \):
Now, let's count the cycle covers of the gadget. First, notice that any cycle cover can only include exactly one outgoing edge from $u$. Thus, once a chain has been chosen, say the $i$th chain, the only way to cover the remaining chains is by their self loops. Thus, we need only count the cycle covers of the $i$th chain and take the sum over all chains $i$, $1 \leq i \leq t$.

Now, let's count the number of ways to cover a given chain. Chain $i$ consists of $k_i$ links, each composed of 3 vertices. To cover the $j$th link we can either take the bottom edge and the self-loop on the top vertex, or we can take the 2 edges to and from the upper vertex. Thus, given that the outgoing edge from $u$ to link $i$ is included in the cover, there are exactly 2 covers of each link and thus there are $2^{k_i}$ ways to cover the chain. Therefore the total number of cycle covers is $2^{k_1} + 2^{k_2} + \cdots + 2^{k_t}$ which in fact is equal to $w$.

Finally, we construct $G'$ by replacing each edge in $G$ by the described gadget and consider its adjacency matrix $A'$. Each edge $e = (u,v) \in E_G$ of weight $w$ has a corresponding gadget in $G'$ with $w$ cycle covers. Recall that $\text{perm}(A)$ is the sum of the weights of all cycle covers in $G$ where the weight of a cycle cover is the product of its edge weights.

**Claim:** The number of cycle covers of $G'$ is equal to the sum of the weights of the cycle covers of $G$.

Consider a fixed cycle cover of $G$ on some set of $m$ edges with weights $w_1, w_2, \ldots, w_m$. The weight of this cover is $\prod_{i=1}^{m} w_i$. In $G'$ the edge of weight $w_k$ corresponds to a gadget with $w_k$ cycle covers. Thus, the number of ways to cover $G'$ using the gadgets associated with these fixed edges is $\prod_{i=1}^{m} w_i$. Thus, the sum of the weights of the cycle covers of $\text{perm}(A)$. That is, $\text{perm}(A') = \text{perm}(A)$.

Thus, we have shown

\[ \text{PERMANENT} \leq_{\text{poly}} \text{POS-PERMANENT} \leq_{\text{poly}} 0\text{-1 PERMANENT} \]
and this proves Valiant’s Theorem.

4 Combinatorial Interpretations

The determinant of a matrix $T$ can be thought of as the degree of “volume distortion” experienced by region under $T$ acting as a linear transformation. This gives a fairly intuitive geometric meaning of determinant, however there is no equivalent interpretation for permanent. As it turns out, many results on the determinant come from its homomorphic property, which the permanent does not possess. Thus, to get a feel for the meaning of the permanent and determinant we can look more closely at some of their graph theoretic interpretations.

4.1 Counting Spanning Trees

Definition 4.1. Let $G$ be an undirected graph on $n$ vertices. The Laplacian matrix $L$ of $G$ is defined as follows:

$$ L_{ij} = \begin{cases} 
  d_i & \text{if } i = j \\
  -e & \text{where } e \text{ is the number of edges between } i \text{ and } j \\
  0 & \text{otherwise.}
\end{cases} $$

Lemma 4.1. The determinant of a matrix $A$ is the product of $A$’s eigenvalues. That is, $\det A = \prod \lambda$.

Lemma 4.2. Let $L$ be the Laplacian of an undirected graph $G$ on $n$ vertices. Then $\det L = 0$.

Proof. The entries of a given row or column of $L$ must sum to 0. Therefore, multiplying $L$ by the vector of all 1s, $\vec{v}_1$ yields the vector of all 0s $\vec{v}_0$. That is, $L \cdot \vec{v}_1 = \vec{v}_0$ and $\vec{v}_0 = 0 \cdot \vec{v}_1$. Thus $\vec{v}_1$ is an eigenvector of $L$ with corresponding eigenvalue $\lambda = 0$. Hence by Lemma 1.1, $\det(L) = 0$.

Theorem 4.1. Let $G$ be an undirected graph and let $T(G)$ denote the number of spanning trees of $G$ and let $L^{(ii)}$ denote the Laplacian of $G$ with the $i$th row and column deleted. Then,

$$ \forall i, T(G) = \det L^{(ii)}. $$

Proof (see [1, p. 654]). For the base case, let suppose $G$ has 2 vertices connected by $d$ edges. Then

$$ L = \begin{pmatrix} 
  d & -d \\
  -d & d
\end{pmatrix} $$

and $L^{(ii)} = (d)$, giving $\det L^{(ii)} = d$ which is clearly equal to the number of spanning trees of $G$. Also note that for a graph on 1 vertex $L$ is the $0 \times 0$ matrix which has determinant 0 by definition.

Now, let $G$ have $n \geq 2$ vertices and $k > 0$ edges. Suppose that the theorem holds for graphs on one fewer edge or vertex. Choose some vertex $i \in V(G)$. If $\deg(i) = 0$, then $T(G) = 0$ and $L^{(ii)}$ is the complete Laplacian for the graph $G - \{i\}$ and by Lemma 1.3 $\det L^{(ii)} = 0$ and the theorem holds. On the other hand, suppose that there is some edge $e = (i, j)$ connecting $i$ to some vertex $j$. Observe that each spanning tree must either include $e$ or not. With this in mind we consider the following two smaller graphs:

1. $G - e$: The graph $G$ with $e$ removed.
2. $G \cdot e$: The graph $G$ with $i$ and $j$ merged into a single vertex $x$ such that $i, j, e$ are no longer in the graph and any edge with $i$ or $j$ as an endpoint now is incident to $x$. 

9
Now, $T$ is a spanning tree of $G - e$ if and only if $T$ is a spanning tree of $G$ and does not include $e$. Each spanning tree of $G \cdot e$ implicitly includes $e$. Thus, we get the following recursive relation:

$$T(G) = T(G - e) + T(G \cdot e).$$  \hfill (4)

By the inductive hypothesis the theorem holds for $G - e$ and $G \cdot e$. Thus our proof would be complete if we could show

$$\det L^{(ii)}(G) = \det L^{(ii)}(G - e) + \det L^{(ii)}(G \cdot e).$$  \hfill (5)

First, let us reorder the vertices in the Laplacian to get the following:

$$L(G) = \begin{pmatrix} d_i - 1 & -1 & r_i^T \\ -1 & d_j - 1 & r_j^T \\ r_i & r_j & L' \end{pmatrix} \hfill \quad L(G - e) = \begin{pmatrix} d_i - 1 & 0 & r_i^T \\ 0 & d_j - 1 & r_j^T \\ r_i & r_j & L' \end{pmatrix} \hfill \quad L(G \cdot e) = \begin{pmatrix} d_i + d_j & r_i^T + r_j^T \\ r_i + r_j & L' \end{pmatrix}$$

Computing the $L^{(1,1)}$ laplacian for each graph we must show

$$\det \begin{pmatrix} d_j \\ r_j \end{pmatrix} L' = \det \begin{pmatrix} d_j - 1 \\ r_j \end{pmatrix} L' + \det L'.$$  \hfill (6)

We now use the fact that the determinant can be written as a linear combination of its cofactors,

$$\det A = \sum_{k=1}^{n} (-1)^k A_{1,k} \det A^{(1,k)}$$

and so

$$\det L^{(1,1)}(G) = \sum_{k=2}^{n} (-1)^k L^{(1,1)}_{1,k} \det L^{(1,1)+(1,k)} - d_j \det L'$$

while

$$\det L^{(1,1)}(G - e) = \sum_{k=2}^{n} (-1)^k L^{(1,1)}_{1,k} \det L^{(1,1)+(1,k)} - (d_j - 1) \det L'.$$

We observe that $\det L^{(1,1)}(G)$ and $\det L^{(1,1)}(G - e)$ are only off by the value of $\det L'$. Returning to eq. (6) we see that $\det L^{(1,1)}(G \cdot e)$ compensates for this difference. Thus eq. (6) holds. This coupled with eq. (5) proves the result:

$$T(G) = \det L^{(ii)}(G)$$  \hfill (7)
4.2 Counting Perfect Matchings

Now that we have a way of understanding the determinant through of graph theoretic interpretation, let’s consider a similar interpretation of the permanent.

**Theorem 4.2.** Let $G$ be a bipartite graph on $2n$ vertices with partitions $V_1$, $V_2$ such that $\forall (u,v) \in E_G$, $u$ and $v$ are in different partitions and $|V_1| = |V_2| = n$. Let $B$ be a matrix where $B_{ij} = 1$ if there is an edge $(i,j)$ in $G$ such that $i \in V_1$, $j \in V_2$, and $B_{ij} = 0$ otherwise. Then $\text{perm}(B)$ is the number of perfect matchings in $G$.

**Proof (see [1, p. 658]).** Let $G$ be a bipartite graph with $2n$ vertices as described. Define the matrix $B$ as described in the theorem.

Each perfect matching on $G$ corresponds to a permutation $\pi : V_1 \to V_2$. Given a permutation $\pi$, notice that $B_{i,\pi(i)} = 1$ if and only if there is an edge $(i, \pi(i))$ joining a vertex $i \in V_1$ to a vertex $\pi(i) \in V_2$. So $\prod_{i=1}^{n} B_{i,\pi(i)} = 1$ if and only if $(i, \pi(i))$ is a valid edge $\forall i$. Otherwise $\prod_{i=1}^{n} B_{i,\pi(i)} = 0$. Thus, the number of perfect matchings of $G$ is given by

$$\text{perm}B = \sum_{\pi} \prod_{i=1}^{n} B_{i,\pi(i)}.$$  

$\square$

More generally, if we let $G$ be a weighted graph and define the weight of a perfect matching as the product of its edge weights, then $\text{perm}B$ gives the total weight of all perfect matchings (where $B_{ij}$ is the weight of the $(i,j)$ edge).

5 Conclusion

This paper is an introductory survey of the Determinant vs. Permanent problem. I have shown proofs of key theorems regarding the complexity of each problem as well as some combinatorial interpretations of each. Development of algorithms which approximate the permanent is also a large area of current research. Another question is, given a matrix $A$, what is the smallest matrix $B$ such that $\text{perm}(A) = \text{det}(B)$?

The determinant is in $\mathbb{P}$ while the permanent is $\#\mathbb{P}$-hard, even on 0-1 matrices. This is representative of a not well understood phenomena in complexity theory when two similar problems exhibit very different properties when it comes to computing a solution. Due to this, the problem has seen a great deal of interesting research since the mid 20th century extending far beyond the scope of this paper.

References


1If $|V_1| \neq |V_2|$, then the number of perfect matchings is 0.