Lecture 26 Counting Problems and #P

In this lecture we discuss the complexity of counting problems. Instead of just determining whether a solution to a given problem exists, we will be interested in counting the number of different solutions to a given problem. Counting problems are naturally associated with many of the decision problems we have already discussed. The notion of a *witness function* formalizes this association.

Definition 26.1 Let $w : \Sigma^* \to \mathcal{P}(\Gamma^*)$, where $\mathcal{P}(\Gamma^*)$ denotes the power set of Γ^*, and let $x \in \Sigma^*$. We refer to the elements of $w(x)$ as *witnesses* for x. We associate a decision problem $A_w \subseteq \Sigma^*$ with w:

$$A_w = \{x \in \Sigma^* \mid w(x) \neq \emptyset\} .$$

In other words, A_w is the set of strings that have witnesses. □

Example 26.2 Let $x \in \Sigma^*$ be an encoding of a Boolean formula and $y \in \Gamma^*$ an encoding of a truth assignment. If

$$w(x) = \{\text{truth assignments satisfying } x\} ,$$

then

$$A_w = \{\text{satisfiable Boolean formulas}\} .$$

□

It is possible for a counting problem to be harder than the associated decision problem. In order to characterize the additional difficulty of these problems, Valiant [103] proposed a new class of problems called $\#P$. His definition is essentially equivalent to the following.

Definition 26.3 The class $\#P$ is the class of witness functions w such that:

(i) there is a polynomial-time algorithm to determine, for a given x and y, whether $y \in w(x)$;

(ii) there exists a constant $k \in \mathcal{N}$ such that for all $y \in w(x)$, $|y| \leq |x|^k$. (The constant k can depend on w).

□

The following theorem relates counting problems in this new class to their associated decision problems.

Theorem 26.4 *The following relationships hold between witness functions in $\#P$ and decision problems in NP:*

(i) if $w \in \#P$ then $A_w \in NP$;

(ii) if $A \in NP$, then there exists a $w \in \#P$ such that $A = A_w$.

Proof.

(i) Guess a witness $y \in \Gamma^*$ in polynomial time using Definition 26.3(ii) and verify in polynomial time that y is indeed a witness using Definition 26.3(i).

(ii) Let M be a nondeterministic Turing machine accepting A. Take $w(x)$ to be the set of accepting computation paths of M on input x.

□

It is interesting to observe how counting problems v and w are related under the process of reduction. For this purpose, we introduce the notion of *counting reductions* and *parsimonious reductions*.

Definition 26.5 Let

$$\begin{aligned} w : \Sigma^* &\rightarrow \mathcal{P}(\Gamma^*) \\ v : \Pi^* &\rightarrow \mathcal{P}(\Delta^*) \end{aligned}$$

be counting problems. A *polynomial-time many-one counting reduction* from w to v consists of a pair of polynomial-time computable functions

$$\begin{aligned} \sigma &: \Sigma^* \rightarrow \Pi^* \\ \tau &: \mathcal{N} \rightarrow \mathcal{N} \end{aligned}$$

such that

$$|w(x)| \quad = \quad \tau(|v(\sigma(x))|) \ .$$

When such a reduction exists we say that w *reduces to* v. □

Intuitively, if one can easily count the number of witnesses of $v(y)$, then one can easily count the number of witnesses of $w(x)$.

Some reductions preserve the number of solutions to a problem exactly. We call such reductions *parsimonious*. Formally,

Definition 26.6 A counting reduction σ, τ is *parsimonious* if τ is the identity function. □

Example 26.7 Here are a number of examples of parsimonious and non-parsimonious reductions.

- Cook's Theorem is parsimonious in the sense that the number of satisfying assignments to the Boolean formula constructed in the proof of the theorem corresponds exactly to the number of accepting computations of the nondeterministic Turing machine being simulated.
- The reduction Clique $\leq_{\mathrm{m}}^{\mathrm{p}}$ Vertex Cover as presented in a previous lecture is parsimonious: the number of cliques of $G = (V, E)$ of size k is the same as the number of vertex covers of $\overline{G} = (V, \overline{E})$ of size $n - k$.
- The reduction CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3CNFSat as presented in a previous lecture is not parsimonious, but it can easily be made so.
- The reduction 3CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ Clique as presented in a previous lecture is not parsimonious, but again can easily be made so.

□

Example 26.8 We show how the reduction 3CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ Clique can be made parsimonious. Recall that we constructed from a given CNF formula $\mathcal{B}$ a graph G with a vertex for each occurrence of a literal in $\mathcal{B}$ and edges between occurrences of literals in different clauses if the literals were not complementary. The problem with this construction is that one truth assignment might correspond to several cliques. For example, the formula

$$(x \vee y) \wedge (x \vee \overline{y})$$

has two satisfying assignments but three 2-cliques.

We can remedy this by first replacing each clause

$$\cdots \wedge (x \vee y \vee z) \wedge \cdots$$

in $\mathcal{B}$ by the equivalent subformula

$$\cdots \wedge (x\,y\,z \vee x\,y\,\overline{z} \vee x\,\overline{y}\,z \vee x\,\overline{y}\,\overline{z} \vee \overline{x}\,y\,z \vee \overline{x}\,y\,\overline{z} \vee \overline{x}\,\overline{y}\,z) \wedge \cdots$$

with seven terms. The number of satisfying assignments is the same, since the formulas are equivalent. Now we construct G with one vertex for each of the seven terms in each of these subformulas and edges connecting terms xyz and uvw in different clauses if the two terms contain no complementary literals. One can show that there is exactly one clique of size k (the number of clauses) for each satisfying assignment. □

As with the *NP*-complete problems, we can define the class of *#P-complete problems* that represent the hardest problems in the class $\#P$. All the counting problems we have mentioned so far are $\#P$-complete.

One $\#P$-complete problem is that of computing the *permanent* of a matrix. Intuitively, the permanent of an $n \times n$ 0-1 matrix is the number of ways to place n rooks on the matrix so that every rook sits on a 1 and no rook can capture another. Officially,

Definition 26.9 Given an $n \times n$ matrix A (not necessarily 0-1), the *permanent* of A is the quantity

$$\text{perm}\, A \;=\; \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where the σ are permutations of the set $\{1, 2, \ldots, n\}$ and S_n is the set of all such permutations. □

The definition of the permanent of a matrix is very similar to the definition of determinant:

$$\det A \;=\; \sum_{\sigma \in S_n} (-1)^{\text{sign } \sigma} \prod_{i=1}^{n} A_{i,\sigma(i)} \;.$$

The only difference is that the sign[4] of the permutation (even or odd) is included in the determinant. It is thus quite surprising that the permanent should be $\#P$-complete, since the determinant is computable in polynomial time by Gaussian elimination.

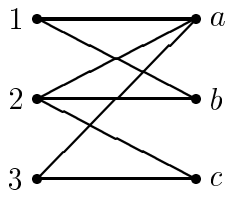
That the permanent is $\#P$-complete is even more surprising in light of its relationship to the bipartite matching problem. Given a bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$, the permanent of the $n \times n$ bipartite adjacency matrix of G gives the number of perfect matchings. Thus, despite the fact that a perfect matching can be found in polynomial time, counting the number of them is as hard as counting the number of satisfying assignments to a Boolean formula.

[4]The *sign* of a permutation of $\{1, 2, \ldots, n\}$ is the number (mod 2) of pairs that are out of order.

Theorem 26.10 (Valiant [103]) *The problem of counting the number of perfect matchings in a bipartite graph is #P-complete.*

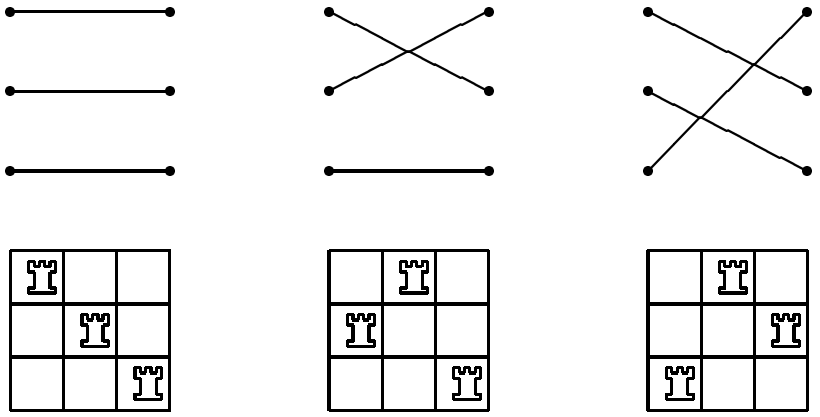
Actually, this problem is #P-complete with respect to a slightly more general reducibility than the one defined in Definition 26.5. We will discuss this later on.

Example 26.11 Consider the following bipartite graph and its associated bipartite adjacency matrix.



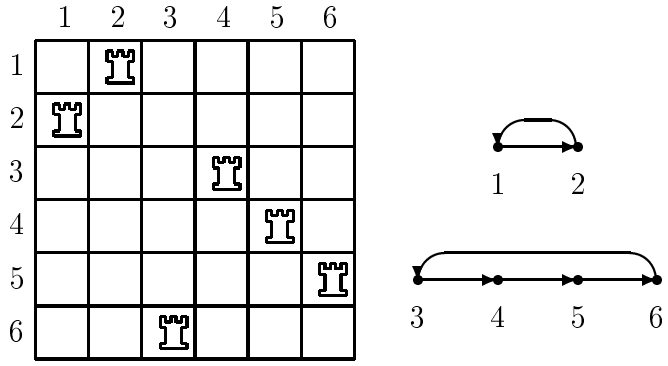
	a	b	c
1	1	1	0
2	1	1	1
3	1	0	1

This graph has exactly three perfect matchings corresponding to the three possible legal rook placements.



□

For non-bipartite directed graphs, we might ask what the permanent of the adjacency matrix represents. Here, a legal rook placement corresponds to a *cycle cover*, or a collection of vertex-disjoint cycles containing all vertices. The permanent thus computes the number of cycle covers. The picture below illustrates the relationship between cycle covers and permutations corresponding to nonzero terms in the definition of the permanent.


1 2 3 4 5 6
1
2
3
4
5
6
1 2
3 4 5 6

Lecture 27 Counting Bipartite Matchings

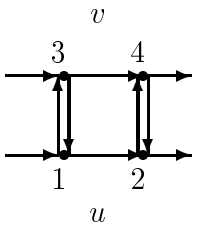
In this lecture we prove that computing the number of perfect matchings in a bipartite graph is $\#P$ complete, making it at least as difficult as computing the number of satisfying truth assignments for a Boolean expression.

We noted last time that the number of perfect matchings in a bipartite graph with the same number of vertices on each side is equal to the *permanent* of its 0-1 adjacency matrix, given by

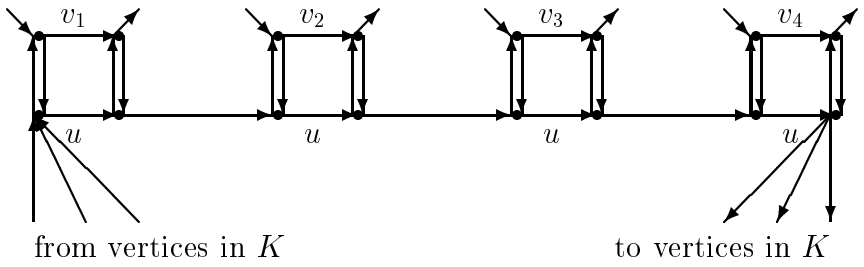
$$\text{perm } A = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where S_n is the set of permutations of the set $\{1, 2, \ldots, n\}$. For a general directed graph with n vertices and its $n \times n$ (nonbipartite) adjacency matrix, the permanent gives the number of cycle covers.

Our proof begins using the same construction that we used to reduce Vertex Cover to Hamiltonian Circuit. Recall that we constructed a graph H from G built from a set of widgets, one for each edge in G.



For each vertex u in G, the sides of the widgets corresponding to u are connected end-to-end to form a *u loop*. The ends of each u loop are connected to each element of a set K of k new vertices as shown; here k is the size of the vertex cover in G we are looking for.



We showed in Lecture 24 that H contains a Hamiltonian circuit if and only if G contains a vertex cover of size k. However, although every Hamiltonian circuit in H determines a unique vertex cover in G, there are in general many different Hamiltonian circuits giving the same vertex cover. How many? For each vertex cover C, each Hamiltonian circuit corresponding to C is determined by the connections between K and the loops corresponding to C, thus we are essentially looking at the number of Hamiltonian circuits in a complete bipartite graph on two sets of k vertices. This number is $k!(k-1)!$, since for each of the $(k-1)!$ cyclic orderings of the vertices in one of the sets of the bipartition, there are $k!$ ways to insert the vertices in the other set so that the two sets alternate.

Similar thinking leads to the conclusion that there are exactly $(k!)^2$ cycle covers in H corresponding to a given vertex cover C in G. This is the number of ways to select two perfect matchings in a complete bipartite graph on two sets of k vertices independently, one to model the edges from K to C and the other to model the edges from C to K.

It would be nice if these cycle covers corresponding to vertex covers included all possible cycle covers in H. We could immediately conclude that by computing the permanent of the adjacency matrix for H and dividing by $(k!)^2$ we could obtain the number of vertex covers in G. Alas, life is not quite this simple; in fact, we are just warming up to the real task.

The problem is that the widgets contain cycles of length two that are included in cycle covers counted by the permanent. Such cycle covers do not correspond to any vertex cover. Let us define a cycle cover to be *bad* if it contains a cycle of length two, *good* otherwise. Good cycle covers must traverse widgets properly, therefore correspond to vertex covers as described above. We can conclude

Theorem 27.1 *The number of good cycle covers in H is $(k!)^2$ times the number of vertex covers of G of size k.*

Thus we would like to count only the good cycle covers of H. Unfortunately, the permanent counts all cycle covers, good and bad. We need to figure out how to prevent bad covers from contributing to the value of the permanent of the adjacency matrix. This leads us to Valiant's Excellent Idea #1: try to assign weights, possibly negative, to the interior edges of widgets so that the product of the edge weights in each good cycle cover is 1, but the product of edge weights in each bad cycle cover is 0. The permanent will then count the number of good cycle covers.

Unfortunately, the task of trying to assign edge weights by trial and error quickly leads nowhere. Even if we throw in more edges—in fact, we might as well consider the complete graph with self-loops—the problem of assigning weights appears hopeless. This brings us to Valiant's Excellent Idea #2: forget the widget itself and look at its adjacency matrix instead, and try to write down the essential properties of the matrix that will give us the desired behavior.

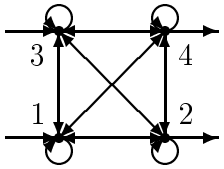
Consider the 4×4 submatrix of the adjacency matrix of H corresponding to one copy W of the widget. There are columns of H corresponding to the "input" vertices 1 and 3 of W and rows of H corresponding to the "output" vertices 2 and 4. In a bad cycle cover involving the two cycles of length two in W, no edge coming into 1 or 3 or leaving 2 or 4 is part of the cycle cover. In terms of the associated legal rook placement, this says that the rooks in columns 1 and 3 and rows 2 and 4 must lie in the submatrix corresponding to W. Moreover, since there are no edges from vertices 1 or 3 to vertices outside of W, *i.e.* all entries in rows 1 and 3 outside of the submatrix W are zero, the rooks in these rows must lie in the submatrix W. This says that there are exactly 4 rooks on the submatrix W, and they form a legal rook placement on W.

When this happens, the remaining rooks must lie in the complementary subgraph W^* of W, *i.e.* that subgraph of H obtained by deleting the rows and columns of W. The sum of all terms in the permanent corresponding to cycle covers containing these two cycles is then given by

$$\operatorname{perm} W \cdot \operatorname{perm} W^* \ .$$

If we could pick weights so that $\operatorname{perm} W = 0$, then the net contribution of all these cycles to the calculation of the permanent would be 0.

With this insight, we proceed to write down all the conditions on the adjacency matrix that guarantee the desired behavior. The intra-widget connections will be as a complete graph with self-loops, and the edges will be weighted. The inter-widget connections will be the same as in H, with all edge weights 1.



Let A denote the 4×4 adjacency matrix of this weighted widget (the weights have yet to be determined). Let $A(i;j)$ denote the submatrix of A obtained by deleting row(s) i and column(s) j.

The desired behavior can be summarized as follows:

$$\begin{aligned} \operatorname{perm} A(4;3) &= \operatorname{perm} A(2;1) = \operatorname{perm} A(2,4;1,3) = 1 \\ \operatorname{perm} A(4;1) &= \operatorname{perm} A(2;3) = \operatorname{perm} A = 0\ . \end{aligned}$$

The first line insures that the net contribution to the permanent of all legal rook placements corresponding to a good cycle cover in H is 1. The three permanents correspond to the three acceptable ways of traversing the widget in a good cycle cover: two zigzag paths and a pair of straight paths. The second line insures that the net contribution of all legal rook placements corresponding to a bad cycle cover is 0. For example, the equation

$$\operatorname{perm} A(4;1) = 0$$

says that the net contribution to the permanent of all cycle covers that have an edge entering the widget at 1 and leaving 4, and no other connections to the outside, is 0.

We do not need to write down any conditions to rule out different numbers of entering and leaving edges; these cases are already taken care of by the rules of rook placement. Essentially, in any cycle cover, the number of edges leaving a subgraph must equal the number of edges entering it.

One can now look for a 4×4 matrix satisfying these constraints. There are many possibilities. One such is

$$A = \begin{bmatrix} 1 & 1 & -1 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

For instance,

$$\begin{aligned} \operatorname{perm} A(4;3) &= \operatorname{perm} \begin{bmatrix} 1 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= 1 \cdot \frac{1}{2} \cdot 1 + 1 \cdot \frac{1}{2} \cdot 1 \\ &= 1 \end{aligned}$$

and

$$\begin{aligned}\operatorname{perm} A(4;1) &= \operatorname{perm} \begin{bmatrix} 1 & -1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= 1 \cdot \frac{1}{2} \cdot 1 - 1 \cdot \frac{1}{2} \cdot 1 \\ &= 0 \ .\end{aligned}$$

The full adjacency matrix B with submatrices A corresponding to these four-node widgets counts 1 for each good cycle cover in H and 0 for each bad cycle cover, thus its permanent is equal to $(k!)^2$ times the number of vertex covers in G.

We have argued that computing the permanent of a matrix containing elements in $\{-1, 0, \frac{1}{2}, 1\}$ is $\#P$-hard, but there is still a ways to go. The next step is to note that

$$\operatorname{perm} 2B = 2^n \cdot \operatorname{perm} B \ ,$$

and this implies that computing the permanent of a matrix with elements in $\{-2, 0, 1, 2\}$ is hard for $\#P$. We now show that this problem reduces to computing the permanents of polynomially many matrices over $\{0, 1\}$. The reduction we use here is somewhat weaker than the one we have been using in that it will require several instances of the $\{0, 1\}$ permanent problem to encode a given instance of the $\{-2, 0, 1, 2\}$ permanent problem, but the reduction still has the property that any fast algorithm for the $\{0, 1\}$ problem would give a fast algorithm for the $\{-2, 0, 1, 2\}$ problem.

Let B be an $n \times n$ matrix over $\{-2, 0, 1, 2\}$. A bound on the absolute value of perm B is given by the case in which each entry of B is 2; then

$$|\operatorname{perm} B| \leq 2^n n! \ .$$

It thus suffices to compute perm B modulo any $N > 2^{n+1} n!$, and from this we will be able to recover the value of perm B.

Let $p_1, p_2, \ldots, p_k$ be the first k primes, where k is the least number such that

$$N = \prod_{i=1}^{k} p_i > 2^{n+1} n! \ .$$

It is not hard to show that $k \leq n + 1$. Moreover, since p_m is $\Theta(m \log m)$ (see [49, p. 10]), we can generate the first k primes in polynomial time using the sieve of Eratosthenes. Before proceeding further, we need the following theorem.

Theorem 27.2 (Chinese Remainder Theorem) *Let $m_1, m_2, \ldots, m_k$ be pairwise relatively prime positive integers, and let $m = \prod_{i=1}^{k} m_i$. Let $\mathcal{Z}_n$*

denote the ring of integers modulo n. The ring $\mathcal{Z}_m$ *and the direct product of rings*

$$\mathcal{Z}_{m_1} \times \mathcal{Z}_{m_2} \times \cdots \times \mathcal{Z}_{m_k}$$

are isomorphic under the function

$$f : \mathcal{Z}_m \rightarrow \mathcal{Z}_{m_1} \times \mathcal{Z}_{m_2} \times \cdots \times \mathcal{Z}_{m_k}$$

given by

$$f(x) = (x \bmod m_1, x \bmod m_2, \ldots, x \bmod m_k) .$$

This just says that the numbers mod m and the k-tuples of numbers mod m_i, $1 \leq i \leq k$, are in one-to-one correspondence, and that arithmetic is preserved under the map f. For example, in the following table, we have compared $\mathcal{Z}_{15}$ to $\mathcal{Z}_3 \times \mathcal{Z}_5$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \bmod 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \bmod 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4

Note that each pair in $\mathcal{Z}_3 \times \mathcal{Z}_5$ occurs exactly once. This is because 3 and 5 are relatively prime. Arithmetic is preserved as well: for example, 4 and 7 correspond to the pairs $(1, 4)$ and $(1, 2)$, respectively; multiplying these pairwise gives the pair $(1, 3)$ (mod 3 and 5, respectively), which occurs under 13; and $4 \times 7 = 28 = 13 \pmod{15}$.

Also, f and f^{-1} are computable in polynomial time. To compute $f(x)$, we just reduce x modulo $m_1, \ldots, m_k$. To compute $f^{-1}(x_1, \ldots, x_k)$, we first compute, for each $1 \leq i \leq k$, integers s and t such that

$$sm_i + t \prod_{\substack{1 \leq j \leq k \\ j \neq i}} m_j = 1$$

and take

$$u_i = t \prod_{\substack{1 \leq j \leq k \\ j \neq i}} m_j .$$

The numbers s and t are available as a byproduct of the Euclidean algorithm. For each $1 \leq i, j \leq k$, $u_i \equiv 1 \bmod m_i$ and $u_i \equiv 0 \bmod m_j$, $i \neq j$. Take

$$f^{-1}(x_1, \ldots, x_k) = x_1 u_1 + \cdots + x_k u_k \bmod m .$$

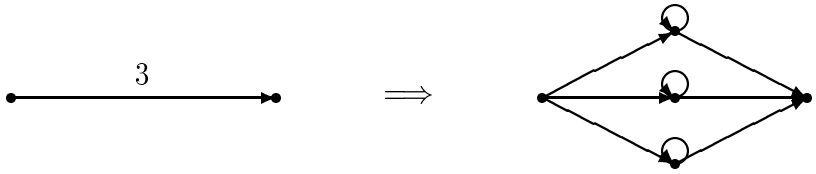
For further details and a proof of the Chinese Remainder Theorem see [3, pp. 289ff.].

Using the Chinese Remainder Theorem, we can compute perm B by computing perm $B \bmod p_i$, $1 \leq i \leq k$. For each i, $1 \leq i \leq k$, we replace all -2 entries in B by $p_i - 2$; modulo p_i, they are the same. We then compute the permanent of this matrix and reduce modulo p_i to get perm $B \bmod p_i$. The advantage of this is that we have now reduced the problem to that of computing the permanents of matrices with small nonnegative entries only.

All that remains is to show how to reduce the computation of the permanent of a matrix over $\{0, 1, 2, p-2\}$ to the problem of computing the permanent of a matrix over $\{0, 1\}$.

Recall the equivalence between the permanent of a matrix and the cycle covers of a directed graph. We must reduce the problem of computing the number of cycle covers of a weighted directed graph with positive integral weights to the problem of computing the number of cycle covers of an unweighted directed graph. This is accomplished by replacing every weighted edge with a subgraph consisting of several new vertices and edges.

The following figure shows this construction for an edge of weight 3.



The above process is repeated for each edge in G. The resulting graph G' is unweighted. Each cycle cover in G involving edges (u_i, v_i) with weights m_i, $1 \leq i \leq n$, is simulated by $m_1 m_2 \cdots m_n$ cycle covers in G', each of weight 1, thus the permanents are the same. Also, G' can be constructed in polynomial time.

This completes the proof that the problem of counting the number of perfect matchings in a bipartite graph (equivalently, counting the number of cycle covers in a directed graph) is $\#P$-complete.