The Class #P, the Permanent, and Counting Linear Extensions

Patrick Niccolai

CS 240

June 5, 2022

1 Background

1.1 Counting Problems

Problems like 3SAT are decision problems. A Turing Machine for 3SAT, given an instance of a 3CNF boolean formula, only needs to determine whether or not the formula has a satisfying assignment. The TM only has to output one bit signifying accept or reject. In this way, TMs that solve decision problems can be viewed as predicates, meaning they are functions that map $\Sigma^* \mapsto \{0, 1\}$.

A counting problem, on the other hand, asks for more than one bit of output. A counting problem asks for the number of solutions to the problem. For example, the counting version of 3SAT is #3SAT. A TM for #3SAT given a boolean formula ϕ would, rather than outputting whether there exists a satisfying assignment for ϕ, output the number of satisfying assignments. So, rather than predicates, counting problems can be viewed as more general functions mapping $\Sigma^* \mapsto \{0, 1\}^*$, where the output is the binary representation of the number of solutions. For notational simplicity, we will refer to counting problems as functions throughout this report, and we will identify a class of TMs by the function or class of functions they compute.

1.2 The Class FP

FP stands for Function Polynomial time. FP is the class of functions which can be computed by a deterministic TM in polynomial time. Essentially, FP is analogous to P, but FP computes functions while P computes predicates. Rather than simply accepting or rejecting an input, an FP TM would write output to a special output tape. For example, a counting problem that could be computed deterministically in polynomial time would be in FP, and the output would be the number of solutions to the problem.

1.3 Oracle TMs

An oracle TM is a TM with the ability to consult an oracle. For our purposes, an oracle can compute polynomial bounded functions. A polynomial bounded function is a function $f : \Sigma^* \mapsto \Sigma^*$ such that $\exists$ polynomial p where $\forall x : |f(x)| \leq p(|x|)$. An oracle TM has two special tapes: a query tape and an answer tape. To consult the oracle, the TM writes x on the query tape. Then, in one computational step, $f(x)$ is written on the answer tape, where f is the function computed by the oracle. Then, the TM can read the answer tape.

If α is a class of TMs (e.g. TMs that compute FP functions), and f is a (polynomial bounded) function, then α^f is the class of functions that can be computed by a TM from α with access to an oracle that computes f.

2 The Class #P

Valiant defines a counting TM as a nondeterministic TM that has an output device which "magically" writes, in binary, the number of accepting computations on a special output tape. The time it takes to write this output does not affect the time complexity of the TM.

#P is the class of functions that can be computed by polynomial time bounded counting TMs. In other words, #P is the class of counting problems whose solutions are the number of paths from the start configuration to an accepting configuration in the configuration graph of some nondeterminstic polynomial time TM, which we will call accepting paths.

#P is an interesting class because the number of accepting paths of a nondeterminstic TM often corresponds to the number of different solutions to the problem. For example, a nondeterminstic TM for 3SAT works by nondeterminstically guessing an assignment to the variables in the formula and then checking if that assignment satisfies the formula. So, the number of accepting paths for the TM would be the number of different assignments which satisfy the formula. Therefore, the problem of counting the number of satisfying assignments, #3SAT, can be computed by counting the number of accepting paths in a nondeterministic TM, so #3SAT $\in$ #P.

2.1 #P-Completeness

#P-completeness is defined in terms of oracle TMs. A function f is #P-hard iff #P $\subseteq$ FP^f. f is #P-complete iff f is #P-hard and $f \in$ #P. The reason for this definition is that if f is #P-complete and it is found that $f \in$ FP, then #P = FP. This is similar to how if an NP-complete problem is found to be in P, then P = NP.

This definition also allows for a notion of reducing any problem in #P to a #P-complete problem. If any problem in #P can be reduced to f in deterministic polynomial time, then f must be #P-complete. This is because for any arbitraty problem a in #P, a deterministic polynomial time TM can perform the reduction from an instance of a to an instance of f and then query an oracle for computing f. Therefore, #P $\subseteq$ FP^f, so f is #P-complete.

2.2 #3SAT is #P-Complete

Valiant notes that generally the counting versions of NP-complete problems are #P-complete. We will show that #3SAT is #P-complete. We have already shown that #3SAT $\in$ #P, so now we need to show that it is #P-hard, i.e. we need to show that #P $\subseteq$ FP$^{\#3SAT}$.

To prove this, we can show that any arbitrary function $a \in$ #P can be reduced to #3SAT in polynomial time. Then, an FP TM for a can do this reduction and then query an oracle for computing #3SAT, meaning that #P $\subseteq$ FP$^{\#3SAT}$ and #3SAT is #P-complete.

Because $a \in$ #P, we know that the decision problem version of a, a_D is in NP. Therefore, using a Cook-Levin reduction we can convert an instance of a_D into an instance of 3SAT. Valiant

claims that with some modifications, the Cook-Levin reduction can ensure that the number is satisfying assignments of the 3SAT formula is equal to the number of accepting paths in a_D's TM. Therefore, using this modified Cook-Levin reduction we can reduce any $a_D \in$ NP to 3SAT in a way that preserves the number of accepting paths. This means that if we could compute #3SAT we could count the number of accepting paths, so we have reduced a to #3SAT, as desired.

3 The Permanent

The equation for the permanent of an $n \times n$ matrix A is:

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

where the summation is over the $n!$ permutations of $(1, 2, ..., n)$. Note that the permanent is very similar to the determinant, which is given by the formula:

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

Despite this similarity, the determinant can be computed efficiently while the permanent cannot. In fact, as we will show later, computing the permanent is #P-complete.

3.1 Graph Interpretation of the Permanent

If the matrix A is considered as the adjacency matrix of a directed graph, then each $\prod_{i=1}^{n} A_{i,\sigma(i)}$ term in Perm(A) is the product of the weights of edges in a cycle cover (a set of vertex-disjoint cycles which covers all vertices of the graph). So, the permanent is the sum of the weights of all the cycle covers. Equivalently, if A is a (0,1)-matrix the permanent is the number of cycle covers.

Another interpretation is that A is the adjacency matrix of a graph where the indices to the rows represent one set of vertices and the indices to the columns represent another set of vertices (this is called a biadjacency matrix). In this case, the graph is bipartite and the permanent is the sum of the weights of perfect matchings in the bipartite graph. Again, if A is a (0,1)-matrix the permanent is the number of perfect matchings.

3.2 Computing the Permanent of a (0,1)-Matrix is in #P

We will refer to the problem of computing the permanent as Perm, and we are interested in Perm of a (0,1)-matrix. Because Perm is equivalent to counting the number of perfect matchings in a bipartite graph, we want to show that determining if a bipartite graph has a perfect matching is in NP (in fact it is in P, but for our purposes it suffices to show that it is in NP). This would mean that there exists a nondeterminstic polynomial time TM that decides whether a bipartite graph has a perfect matching. Furthermore, we want to ensure that the number of accepting paths in this TM is equivalent to the number of perfect matchings. If this is true, then we have shown that Perm is in #P.

The nondeterministic TM functions as you might expect. First, it nondeterministically guesses a set of edges. Then, it checks that the set of edges is indeed a perfect matching. This TM is

polynomial time bounded and it does decide if a bipartite graph has a perfect matching. Also, the number of accepting paths equals the number of perfect matchings. This is because each path in the configuration graph corresponds to a different nondeterministic choice of a set of edges, and every perfect matching will have its own path to an accepting configuration. Therefore, counting perfect matchings in a bipartite graph is in #P and so Perm is in #P.

3.3 Computing the Permanent of a (0,1)-Matrix is #P-Complete

To give a brief overview of the proof, we will take a boolean formula ϕ and use it to create a graph where the permanent of the adjacency matrix depends on the number of satisfying assignments of ϕ. Then we will use a "modulo many primes" trick to reduce Perm of this adjacency matrix to Perm of several (0,1)-matrices, thus reducing #3SAT to Perm of (0,1)-matrices, showing that Perm of a (0,1)-matrix is #P-complete.

We have shown that Perm is in #P, so now we need to show that it is #P-hard. To do this, we will show a reduction from #3SAT to Perm. We will make use of the graph interpretation of the permanent by showing how to reduce a 3CNF boolean formula ϕ to a graph G such that the adjacency matrix A of G has the property that

$$\text{Perm}(A) = 4^{t(\phi)} \cdot s(\phi)$$

where $t(\phi)$ equals twice the number of occurrences of literals in ϕ minus the number of clauses, and $s(\phi)$ equals the number of satisfying assignment. If we can construct this graph, then we will have reduced #3SAT to Perm, because given an oracle for Perm, an FP TM could get Perm(A) and then solve for $s(\phi)$ in polynomial time. Therefore, #3SAT $\subseteq \text{FP}^{\text{Perm}}$ and so Perm is #P-complete.

The details of constructing this graph can be found in [Val74][1]. Essentially, the graph is comprised of three types of subgraphs: a *track* for every variable in ϕ, an *interchange* for each clause in ϕ, and a *junction* wherever a track and an interchange meet. Valiant claims that the crucial part of this graph is the construction of the junctions, which are 4 four vertex subgraphs given by the following adjacency matrix:

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

What is important about this graph is that it is constructed so that satisfying assignments to ϕ will correspond to a positive weight cycle cover (thus contributing to the permanent), and unsatisfying assignments to ϕ will be canceled out by negative weight edges. As you can see from the above matrix, the entries of the adjacency matrix of junctions, and in fact of adjacency matrix of the entire graph, are in $\{-1, 0, 1, 2, 3\}$. So, we still need further reductions to get a (0,1)-matrix.

To handle edges with weight $k > 1$, we can replace that edge with a loop which itself contains k self-loops. Doing this will remove all entries > 1 in the adjacency matrix while not changing permanent. Now we have a matrix with entries in $\{-1, 0, 1\}$.

Note that the permanent of this matrix is between $-r!$ and $r!$, where r is the number of rows

[1] If you can find it. Valiant cites his own unpublished manuscript, which I couldn't find online.

(or columns) of the matrix. Using the Chinese Remainder Theorem, we can compute Perm(A) by computing Perm(A) $\mod p_i$ for each $p_i \in \{p_0, ..., p_t\}$ which is a set of primes whose product is $\geq 2 \cdot r!$. Also, note that when computing Perm(A) $\mod p_i$, we can replace the -1 entries with $p_i - 1$. Because this is a positive number, we can use the technique above to reduce this new matrix to a (0,1)-matrix. So, now we have reduced #3SAT to Perm of (0,1)-matrices, completing the proof that Perm of a (0,1)-matrix is #P-complete.

4 Counting Linear Extensions

4.1 Posets and Linear Extension

A partially ordered set (poset) is a set which has a transitive relation $<$ among some (or all) of its elements. Two elements x and y are said to be comparable if there is a relation between them (i.e. $x < y$ or $y < x$) in the poset. x and y are said to be incomparable if there is no relation between them in the poset. A linear extension of a poset P is a total ordering $\prec$ of P such that whenever $x < y$ in the poset, $x \prec y$ in the linear extension. Another way to look at a linear extension is that it is a bijection $\lambda : P \mapsto \{0, 1, ..., n\}$ with the property that whenever $x < y$, $\lambda(x) < \lambda(y)$. Essentially, a linear extension extends a partial ordering into a total ordering. We can let $\Lambda(P)$ be the set of all linear extensions of a poset P, and let $N(P)$ be the size of $\Lambda(P)$.

Note that one poset can have many linear extensions. Whenever $x < y$ is *not* a relation in the poset, a linear extension could have $x \prec y$ or $y \prec x$. A large poset with a small number of $<$ relations could have very many linear extensions, and as we will show, determining the number of linear extensions for a given poset is #P-complete.

Posets also have antichains. An antichain is a subset of the elements of a poset where all pairs of elements in the antichain are incomparable.

A poset can be visualized by a directed graph, where the elements of the poset are the vertices of the graph. Let v_x and v_y be the vertices corresponds to elements x and y of the poset, respectively. If $x < y$ is a relation in the poset, then there will be an edge from v_x to v_y in the graph. We can also say that v_y is above v_x. We will use this notion of a poset as a graph in the proof that counting linear extensions is #P-complete.

4.2 Counting Linear Extensions is in #P

To show that the problem of counting linear extensions, which we will call #LINEXT, is in #P, we can show that the decision problem version of #LINEXT, deciding if a poset has a linear extension, is in NP. The nondeterministic TM works by nondeterministically guessing a total ordering and then checking that this ordering does not contradict any $<$ relations in the poset. Each path in the configuration graph of this TM corresponds to a total ordering, and each total ordering that is a valid linear extension would have one accepting path. Therefore, #LINEXT is in #P.

4.3 Counting Linear Extensions is #P-Complete

This proof is similar to the proof of Perm being #P-complete. To give a brief overview, we use boolean formula ϕ to create a poset Q_{p_i} for each prime p_i in a set of primes where the number of linear extensions of Q_{p_i} depends on the number of satisfying assignments to ϕ mod p_i. Using

the same "modulo many primes" trick as before, this will allow us to get the number of satisfying assignments to ϕ, thus reducing #3SAT to #LINEXT.

We have shown that #LINEXT is in #P, so now we need to show that it is #P-hard. To do this, we do a direct proof, i.e. we show that with access to an oracle that computes #LINEXT, a deterministic polynomial time TM can compute #3SAT, thus showing that $\#\text{P} \subseteq \text{FP}^{\#\text{LINEXT}}$. Similar to the proof of the permanent being #P-complete, we want to take a boolean formula and create a poset from it such that the number of linear extensions of the poset depends on the number of satisfying assignments to the boolean formula. Then, if we can compute the number of linear extensions of the poset we can also compute the number of satisfying assignments to the boolean formula.

Let ϕ be a boolean formula with n variables and m clauses. We will construct a poset P_ϕ of size $7m + n$. In the graph of P_ϕ, there is a vertex for every variable of ϕ and 7 vertices for every clause. If x, y, and z are three variables in a clause, then v_x, v_y, and v_z will be vertices in the graph. Above these vertices, there will be 7 vertices corresponding to each nonempty subset of $\{v_x, v_y, v_z\}$. i.e. above v_x will be 4 vertices, corresponding to $\{v_x\}, \{v_x, v_y\}, \{v_x, v_z\}$, and $\{v_x, v_y, v_z\}$. Using our #LINEXT oracle, we can calculate the number of linear extensions of P_ϕ, let L_ϕ be that number.

Next, we find a set of primes $\{p_0, ..., p_t\}$ of primes between $7n + m$ and $(7n + m)^2$ such that no p_i in this set divides L_ϕ and the product of the primes is $\geq 2^n$. For each p_i, we can construct a poset Q_{p_i}. The graph of Q_{p_i} has two vertices for each variable x, one representing x and one representing $\overline{x}$. The graph also has 8 vertices for each clause. If a clause is made up of x, y, and z, then there will be 8 vertices for that clause, each consisting of one element from each of $\{x, \overline{x}\}, \{y, \overline{y}\}, \{z, \overline{z}\}$. The graph also has two special vertices, a and b. The vertex that corresponds to the clause how it actually appears (i.e. if the clause in ϕ is $(x \vee \overline{y} \vee z)$ then we are talking about the unique vertex which is above $x, \overline{y}, z$) is above b while the other 7 vertices are incomparable with b. Also, there is an antichain for each variable. All of these antichains are under a, and the antichain corresponding to the variable x is also under the two vertices corresponding to x and $\overline{x}$. Finally, there is an antichain for each clause. All of these antichains are under b, and the antichain corresponding to clause c is also under the 8 vertices corresponding to c [2].

What is important about these Q_{p_i}'s is that $N(Q_{p_i}) = N_0 \cdot s(\phi) \mod p_i$, where $N_0 =$

$$\frac{(p_i(n+1)-1)!}{p_i^n} \cdot \frac{(p_i(m+1)-1)!}{p_i^m} \cdot L_\phi$$

Because p_i, n, m, and L_ϕ are known values, N_0 can be calculated and therefore, with our oracle that counts linear extensions, we can get $s(\phi) \mod p_i$ for each p_i. Because the product of the p_i's is $\geq 2^n$ and the maximum number of linear extensions is 2^n, we can use the Chinese Remainder Theorem to get $s(\phi)$. Therefore, we have reduced #3SAT to #LINEXT.

Brightwell and Winkler note that this is an interesting result. Normally, decision problems that are in NP have counting problems that are #P-complete. Sometimes, decisions problems in P have counting problems that are #P-complete (e.g. the decision problem of checking if a bipartite graph has a perfect matching is in P. We have shown that counting perfect matchings in a bipartite

[2] Look at Figures 1 and 2 in [BW91] to see images of these constructions.

graph is equivalent to computing the permanent, which is #P-complete). Interestingly, the decision problem of deciding if a poset has a partial ordering is trivial, it is always true, but the counting problem is still #P-complete.

5 Approximating Counting Linear Extensions

#LINEXT has the property that it is self reducible. This means that if we are trying to count linear extensions of a poset P, we can create two posets P' and P'' such that

$$\#\text{LINEXT}(P) = \#\text{LINEXT}(P') + \#\text{LINEXT}(P'')$$

where P' and P'' are created by choosing an incomparable pair in P and making it comparable. i.e. if $x < y$ is *not* a relation in P, then P' would be the same poset as P but with the relation $x < y$ added, and similarly P'' would be P with $y < x$ added.

Now, if we could uniformly sample the space of linear extensions of P, we could get a number $p := \Pr[\text{our sample from } P \text{ is also in } P']$. Then, we could approximate

$$\#\text{LINEXT}(P) \approx \frac{\#\text{LINEXT}(P')}{p}$$

So, if we could efficiently sample uniformly from the space of linear extensions, then we could efficiently approximate #LINEXT. In fact, we can get a *fully polynomial randomized approximation scheme* (*fpras*) for #LINEXT. The means that we can approximate the number of linear extensions to within a factor of ε for any ε we want, where the running time of the approximation algorithm is polynomial in the size of the input and ε^{-1}. The only remaining problem is: how can we sample the space of linear extensions in polynomial time if we can't even count how large this space is?

5.1 Markov Chains

A Markov Chain is a stochastic process, for our purposes it can be though of as a random walk on a graph whose vertices represent the linear extensions of a poset, and there is an edge between two vertices if their corresponding linear extensions are 1 "swap" of an element apart. This graph is called the state space of linear extensions of the poset. If we take a random walk on this graph, after a while we will be at a random linear extension, which we can use as a random sample. However, if the Markov Chain has not run for long enough, the distribution of our samples will not be uniform. The mixing time of a Markov Chain is how many steps it needs to run in order to reach the uniform distribution. So, if we can find a Markov Chain with a low mixing time, we will be able to efficiently uniformly sample linear extensions.

5.2 Mixing Times

At the time of Brightwell and Winkler's paper (published in 1991), the fastest mixing time for a Markov Chain for sampling linear extensions was $O(n^6 \log n \log \varepsilon^{-1})$, from Karzanov and Karzanov [KK91]. In a paper by Bubley and Dyer (published in 1999) [BD99] , they developed a new Markov Chain for counting linear extensions which has a mixing time of $O(n^3 \log n\varepsilon^{-1})$, a large improvement.

5.3 Bubley and Dyer's Markov Chain

The chain works by picking $p \in \{1, 2, ..., n-1\}$ according to a distribution h and $c \in \{0, 1\}$ uniformly at random. If $c = 0$, then the chain will stay in the same state. Otherwise, if swapping elements p and $p+1$ in the poset which corresponds to the vertex where the chain currently is does not violate a relation in the poset, then do that swap. This will move the chain to a new poset, which is a neighbor of the previous poset in the state space. If swapping p and $p+1$ is not a valid move because $p < p+1$ is a relation in the poset, then the Markov Chain will not move.

The probability distribution Bubley and Dyer choose if defined by

$$h(i) = \frac{i(n-i)}{K}$$

where K is a normalizing constant.

To show that this chain will reach the uniform distribution in $O(n^3 \log n\varepsilon^{-1})$ steps, Bubley and Dyer used the path coupling technique. Essentially, they consider two Markov Chains, one starting in an arbitrary position and one starting already in the uniform distribution. By definition, the chain that is in the uniform distribution will not leave the uniform distribution. Then, they show that by $O(n^3 \log n\varepsilon^{-1})$ steps, the first chain will have merged with the second chain (meaning that the two chains will be at the same state) with high probability. Therefore, this shows that by $O(n^3 \log n\varepsilon^{-1})$ steps, the first chain will be in the uniform distribution with high probability.

5.4 Approximating other #P Problems

This efficient algorithm for approximating #LINEXT is good news for approximating other problems in #P. Because #LINEXT is #P-complete, any problem in #P can be reduced to #LINEXT in polynomial time, and then the algorithm for approximating #LINEXT can be used to approximate the original problem. However, while any problem can be reduced to #LINEXT in polynomial time, we do not know that the polynomial will be small. So, in many cases it may be better to develop different Markov Chains for sampling the state space of those problems. Because #LINEXT is #P-complete, it is in a sense among the hardest problems in #P. So, the existence of a Markov Chain with a low mixing time means that hopefully Markov Chains with similar or ever lower mixing times can be discovered for other #P problems.

References

[BD99] R. Bubley and M. Dyer. "Faster random generation of linear extensions". In: *Discrete Mathematics* 201.1 (1999), pp. 81–88. DOI: https://doi.org/10.1016/S0012-365X(98)00333-1.

[BW91] G. Brightwell and P. Winkler. "Counting Linear Extensions". In: *Order* 8 (1991), pp. 225–242. DOI: https://doi.org/10.1007/BF00383444.

[KK91] A. Karzanov and L. Khachiyan. "On the conductance of order Markov chains". In: *Order* 8 (1991), pp. 7–15. DOI: https://doi.org/10.1007/BF00385809.

[Val74] L. G. Valiant. "A reduction from satisfiability to Hamiltonian circuits that preserves the number of solutions". Unpublished manuscript, Leeds. 1974.

[Val79] L. G. Valiant. "The complexity of computing the permanent". In: *Theoretical Computer Science* 8.2 (1979), pp. 189–201. DOI: https://doi.org/10.1016/0304-3975(79)90044-6.