**6.890: Algorithmic Lower Bounds: Fun With Hardness Proofs** Fall 2014

Lecture 15 — October 23, 2014

*Prof. Erik Demaine* *Scribes: Hugo Akitaya, Aahlad Gogineni, Rajan Udwani*

# 1 Recap

We have seen many decision problems when we are concerned whether a solution exists. In approximation problems we had to define a stronger version of NP-style problems which are the optimization problem (NPO) when we want to find the best solution (that maximizes or minimizes some parameter). This class will cover counting problems and, as we did in the past, we will define another version of NP-style problems: NP search problems. Now we are concerned in finding **a** solution for some instance of a problem. From this concept, we define counting problems.

# 2 Overview

NP search problems are a version of NP-style problems whose goal is to find one solution (any solution) to a given NP problem. If the answer to the decision problem is *yes*, than at least one valid solution exists. We can convert a searching problem into a counting problem (that outputs an integer) that counts the number of solutions for a given instance of the search problem. If the original problem was denoted by $A$, we refer as $\#A$ the counting version of $A$. In general, counting problems are harder than its decision version since we can conclude in constant time what is the answer for the decision version (is the output zero or not?). Indeed there are problems whose decision version are polynomial while the counting problem is hard.

A motivation for counting problems comes from puzzle design. We usually want to know if there is a unique solution for a puzzle. $\#P$ is the class of all counting problems. From the computation perspective, we can also define as $\#P = \{$problems solved by polynomial-time nondeterministic counting algorithms$\}$. A NP decision nondeterministic algorithm, when there is branching the algorithm guesses and go to any path that leads to a *yes*. In the counting case, the computer runs all the branches, count the number of *yes* and returns that number. (See Negative Results on Counting by Valiant [4])

A $\#P$-hard problem is a problem as hard as any problem in $\#P$. We can reduce all problems in $\#P$ to a $\#P$-hard problem.

# 3 Reductions

The reductions to prove $\#P$-hardness are usually multicall (Cook-style) reductions. This is a general approach for FNP (NP-style functions whose output is a value and its decision version is in NP). Since the output is a number, in our case, we are also allowed to manipulate the number in the end after a call as well as making multicalls (polynomial number of calls).

### 3.1 Parsimonious reductions

This is a more familiar style of reduction that allows us to prove $\#P$-hardness often without multicalls. As usual, we convert an instance $x$ of problem $A$ into an instance $x'$ of $B$ by a function $f$ that is computable in polynomial time. The difference is that we will require that the number of solutions of $A$ to instance $x$ equals the number of solutions of $B$ to the instance $x'$. This is a stronger notion than a normal NP-style reduction. If $\#A$ is $\#P$-hard, this reduction will imply that $\#B$ is also $\#P$-hard.

A lot of NP reductions that we already saw in this course follows this extra rules and can be used as Parsimonious reductions.

### 3.2 c-monious reduction

This name was made up by Erik. By its etymology the word parsimonious means "little money", so "c-monious" would mean a little more money. Something like "c-money". In this style of reductions we will require that the number of solutions of $x'$ equals $c$ times the number of solutions of $x$. In other words, $c \cdot \#A$ solutions to $x = \#B$ solutions to $x'$. The number $c$ must be a constant (different than zero) to each instance, not dependent of the instance but can be dependent on $n$. From $\#P$ perspective, this is just as good.

## 4 Familiar problems that are (or not) #P-complete

In this section, we will look on examples of past NP reductions and see if they are Parsimonious or not.

### 4.1 #3SAT(-3)

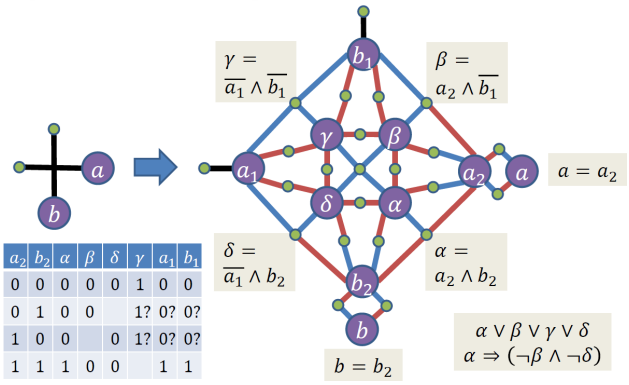
#3SAT is hard. This proof can be done by a reduction from #SAT. #3SAT-3 can also be shown $\#P$-hard by replacing each variable by a cycle.

### 4.2 Planar #3SAT

Planar #3SAT is hard. Figure 1 is from lecture 7. If you see all the cases for this gadget you will conclude that all the variables are forced to assume specific values that depend only of $a$ and $b$. In other words, we can deterministically determine the values for all the variables in the gadget from the values of $a$ and $b$. This means you preserve the number of solutions. We can use exactly the same reduction as lecture 7 to prove that Planar #3SAT is $\#P$-hard.

### 4.3 Planar Monotone Rectilinear #3SAT

The previously seen reduction will also work in this case. Figure 2 shows the reduction. All the variables created to substitute $x_i$ ($a$ and $b$ in the figure) are forced and can be deterministically determined by the value of $x_i$. Therefore, it will preserve the number of solutions.




| $a_2$ | $b_2$ | $\alpha$ | $\beta$ | $\delta$ | $\gamma$ | $a_1$ | $b_1$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | | 1? | 0? | 0? |
| 1 | 0 | 0 | | 0 | 1? | 0? | 0? |
| 1 | 1 | 1 | 0 | 0 | | 1 | 1 |

Figure 1: Figure of the crossover gadget of the reduction from 3SAT to Planar 3SAT.

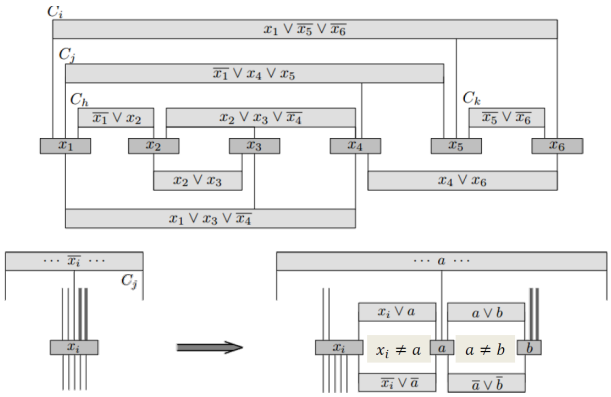



Figure 2: Reduction from Planar #3SAT to Planar Monotone Rectilinear #3SAT.

### 4.4 Planar Positive Rectilinear #1-in-3SAT

The reduction from Planar #3SAT shown in Figure 3 does not work. There is exactly one case when we can set the variables of the original instance and cannot force the internal variables to an specific value. The figure shows that there are two possible solutions of the #1-in-3SAT instance that correspond to the same solution of the #3SAT instance. If every solution of the #3SAT instance had 2 correspondent #1-in-3SAT solutions we would have a c-monious reduction, but this irregularity makes this reduction not good enough for $\#P$-hardness.

Luckily, there was stronger reduction from Planar Rectilinear 3SAT to Planar Positive Rectilinear

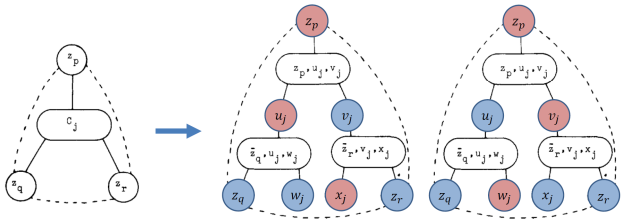


Figure 3: Reduction from Planar 3SAT to Planar 1-in-3SAT.

1-in-3SAT that does work (slide from lecture 7). In this proof, all the variables are forced. The number of solutions is conserved and we have a Parsimonious reduction.

### 4.5 Shakashaka

The reduction from Planar 3SAT in lecture 7 will work as a Parsimonious reduction. Once you decide the value of the variables, everything is forced and the number of solutions for the instance of Shakashaka will be exactly equal to the number of solutions of the 3SAT instance. (This proof is from the paper by Demaine, Okamoto, Uehara and Uno in [1])

### 4.6 Hamiltonian cycle

This is an example of reduction that does not work in the #P perspective. We had two proofs for Hamiltonian Cycle and neither of them are Parsimonious. Figure 4 shows that the clause gadget admits more than one solution and therefore is not Parsimonious.

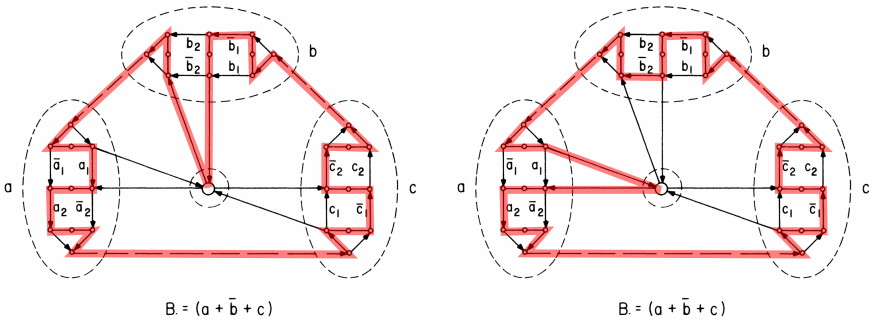



Figure 4: Clause gadget of the reduction to Planar Hamiltonian Cycle. The gadget admits more than one solution.

The other reduction is from lecture 8 was to Planar Directed Max-Degree-3. The same issue occurs in this reduction. When multiple variables satisfy a clause any of them can grab the vertices in the clause gadget. Since we can sometimes have 1, 2 or 3 possible solutions that correspond to the same 3SAT solution, we don't have a parsimonious (or c-monious) reduction.

However, this proof is useful to make a parsimonious reduction. We can use the XOR-gadget of this old proof into a new parsimonious reduction ([Sato, 2002]). This gadget admits only one solution and forces that only (and exactly) one edge connected by the gadget is used in the cycle.

Three new gadget were created for this reduction. They are shown in Figure 5 and are: OR (that requires that one or two of the edges are used in the cycle), implication (that forces an edge to be used if the other edge is used) and a 3-way OR gate (that is used in the clause). The difference of this clause gadget with a 3-way or to the previous reduction is that the new gadget has all its edges forced admitting only one cycle per solution of the original 3SAT instance. Consequently, this new reduction is Parsimonious and prove that finding the number of Hamiltonian cycles in a planar max degree-3 graph is #P-hard.

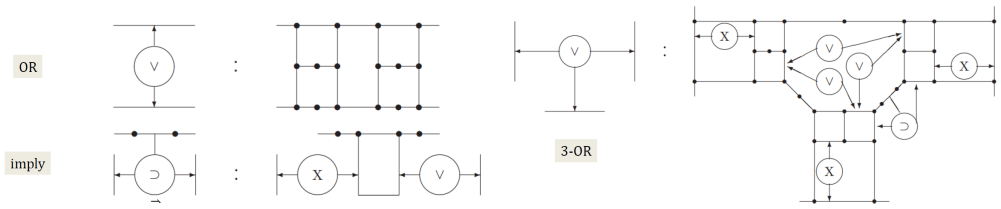
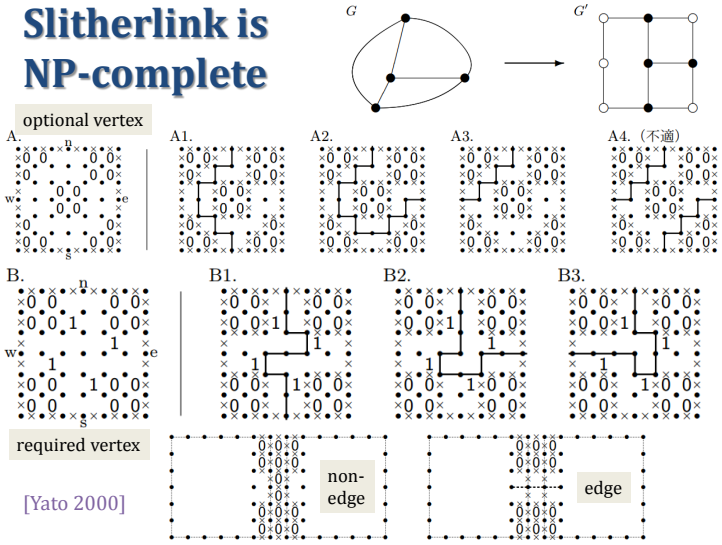



Figure 5: New gadgets for the reduction to Hamiltonian Cycle.

### 4.7 Slitherlink

Here we can't use the reduction from Hamiltonicity problems on grid graphs because we did not see any #-P hardness results for the same but we can use the alternative reduction from Hamiltonian cycles on planar max degree 3 graphs (which was shown to be #-P hard above). Observe the figure below and notice that for any Hamiltonian cycle (i.e. order in wich we visit the vertices) in the planar graph, there is a unique way to solve the puzzle by visitng the 'required' vertices in the corresponding order and given this order there is exactly one way to use the ooptional vertices to complete the cycle. Hence the reduction is parsimonious and we get #-P hardness for Slitherlink.



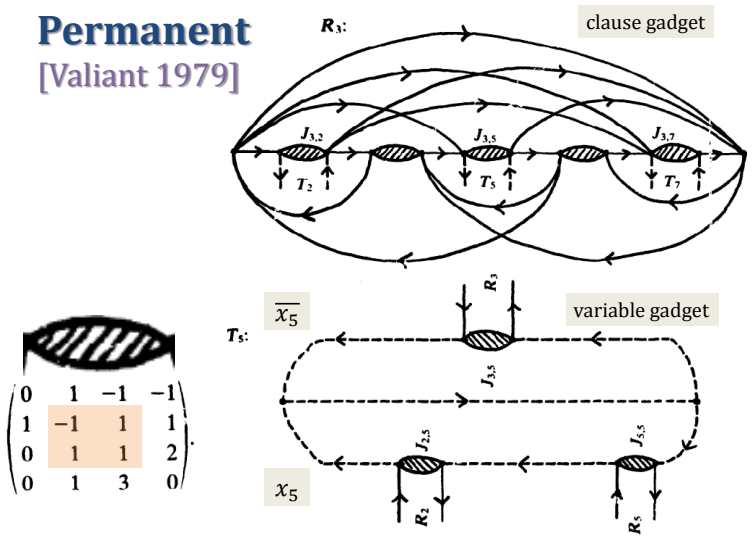

## 5 Other problems

### 5.1 Permanent

**Determinant**: One way to define the determinant of an $nxn$ matrix $A = (a_{i,j})$ is $|A| = \sum_{\pi}(-1)^{sign(\pi)}\Pi_{i=1}^{n}a_{i,\pi(i)}$ where the sum is over all permutations $\pi$of the set $\{1, 2, \ldots, n\}$ and $sign(.)$ of a permutation is a 0/1 value given by the parity of the number of inversions $mod$ 2. Note that even though this definition has an exponential (in $n$) number of terms, computing the determinant is in P.

**Permanent**: is of a matrix $A$ as above is defined as $perm(A) = \sum_{\pi} \Pi_{i=1}^{n} a_{i,\pi(i)}$ i.e. the unsigned sum over permutations. It can also be interpreted as the the sum (over all cycle covers of a directed graph) of the product of weights of edges in a given cycle cover, where the weight of an edge $(i, j)$ is $w(i, j) = a_{i,j}$ and a cycle cover is a set of vertex disjoint directed cycles that cover all vertices of the graph ($|V| = n$).

The problem of finding the Permanent of a matrix is #-P complete. (See paper by Valiant in [2]) Hardness is shown by a $c$-monious reduction from #-3SAT. Look at the figure below




All edges shown in the gadgets are weight one. The shaded structure used in the gadgets is a graph with weight matrix (X) as shown in figure. Notice that the Permanent of the matrix is 0 hence in any nonzero weight cycle cover for the whole graph all such structures ust be a part of a bigger cycle that enters and leaves at some point (else there would be a 0 in the product).

Also, although not explicitly shown, a cycle can enter/exit these structures only at the structures' nodes that correspond to ro/column 1 and row/column 4. If a cycle enters and leaves at the same node (1 or 4) then we again get a 0 in the product due to the remaining part of the structure because the Permanent of X-row/column 1 and X-row/column 4 is also 0. Hence cycles must enter every node and leave at dstinct points for all structures.

Also, X-row/columns 1,4 has permanent 0 as well, hence the cycle entering and exiting the structure at nodes 1,4 must go through at least one more internal node as well to have a non-zero product weight.

Now the premanent of matrix X-row 1, column 4 and of X-row4, column 1 is 4 hence we get a factor 4 in the weight product of a cycle cover due to each structure.

In summary the structures act as forced edges in variable and clause gadgets. In the variable gadget this forces the choice of either $x$ or $\bar{x}$ and in the clause gadget if none of the literal structures have been covered by external edges $(T_2, T_5, T_7)$ then we must traverse all 5 structures in the clause in a consecutive manner but having done that there is no way to finish a cycle, hence atleast one of the structures has to be covered by an external cycle, which corresponds to that literal being 1.

Now note that for every solution to the 3SAT instance there are exactly $4^5$ possible cycle covers

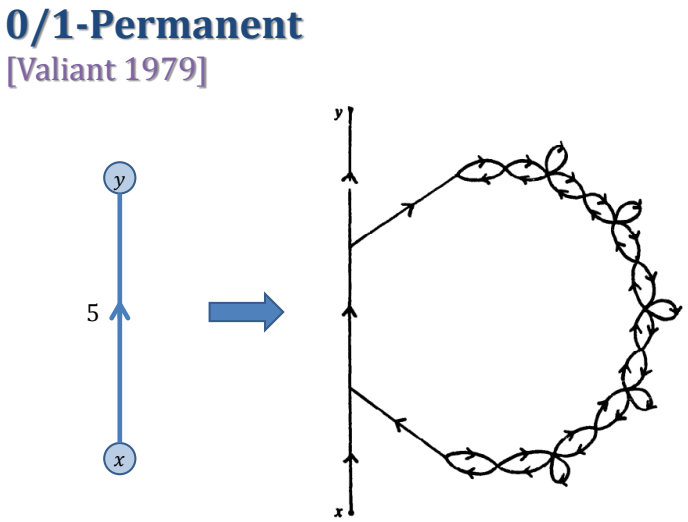
for every clause (because of 5 structures in a clause gadget each coverable 4 different ways), given the variable assignment of the 3SAT solution. Hence the reduction is a $c$-monious reduction with $c = 4^{5(\#clauses)}$.

### 5.2 Permanent mod $r$

This is also #-P hard by a multicall reduction from the previosu problem, because given a polytime algorithm for the *mod* $r$ problem one can substitute $r = 2, 3, 5, 7, 11 \ldots$ until product is $> M^n n!$ where $M$ is the largest albsolute entry in the matrix and $perm(A) \leq M^n n!$. This requires $O(n \log M + n \log n)$ many calls which is polynomial in the size of the input. Hence using the results of these calls and the Chinese Remainder Theorem we can get the Permanent in polytime. Hence the problem is as hard as calculating the Permanent. (See [2])

### 5.3 0/1-Permanent mod $r$

We show it is $\#P$-hard using a parsimonious reduction from Permanent mod r. In any instance of Permanent mod $r$, the edge weights can be represented as non-negative integers $0...r-1$. We replace each edge of weight $k$ with a gadget with $k$ loops. The figure below shows the gadget for an edge of weight 5. (See [2])




If $(x, y)$ is not covered by a cycle in the instance of Permanent mod $r$ there is exactly one way to cover the edges. If $(x, y)$ is covered in a cycle there are exactly $k$ solutions. (each using a different self-loop)

Therefore 0/1-Permanent mod r is $\#P$-hard.

### 5.4 0/1-Permanent

We reduce from 0/1-Permanent mod $r$ using a one-call reduction. Given an instance of 0/1-Permanent mod $r$, if we can find the Permanent of the input we can find the Permanent mod $r$ by

taking the modulus of the output w.r.t $r$.

(This was proved in [2])

Note:

- The permanent of a 0/1 matrix is equal to the number of vertex-disjoint cycle covers in the corresponding directed graph.
- The permanent is also equal to the number of perfect matchings in the bipartite graph of rows and columns of the given matrix. (where the partite sets correspond to the set of rows and set of columns and there is an edge between row $i$ and column $j$ if and only if the element $(i, j) = 1$ )

### 5.5 Bipartite #Maximal Matchings

We use a one-call reduction from Bipartite #Perfect Matchings. Given a bipartite graph $G = (V, E)$ replace each vertex $v$ with $n = |V|$ copies of it $v_1, v_2...v_n$ and replace each edge $(u, v)$ with a biclique (complete bipartite graph) on the sets $\{v_1, v_2...v_n\}$ and $\{u_1, u_2...u_n\}$. For each matching of size $i$ in the original graph there are $n!^i$ distinct matchings of size $ni$ in the transformed graph. In addition maximality of matchings is preserved.

Therefore number of maximal matchings is $\sum_{i=0}^{n/2}$(number of matchings of size $i$)$(n!)^i$. From this number we can extract the number of perfect matchings which is the term in the summation corresponding to $i = n/2$.

Therefore this problem is $\#P$-hard.

### 5.6 Bipartite #Matchings

We describe a multicall reduction from Bipartite #Perfect Matchings. Given the graph $G = (V, E)$. Let $G_k$ be the graph created by adding $k$ adjacent leaves to each vertex in $V$. Suppose there are $M_r$ matchings of size $n - r$ in $G$. They are contained in the $M_r(k + 1)^r$ matchings obtained by adding some number of the new edges. Therefore number of matchings in $G_k = \sum_{r=0}^{n/2} M_r(k+1)^r$ We can evaluate this polynomial in $k$ for $k = 1, 2 \ldots n/2 + 1$ to extract the coefficients $M_0, M_1, \ldots$. Here $M_0$ is the number of perfect matchings.

Therefore this problem is $\#P$-hard.

(This was proved in [3])

### 5.7 Positive #2SAT

We describe a parsimonious reduction from bipartite #matchings. Given a graph $G = (V, E)$, convert each edge into a variable. It is true if it is not in a matching. We convert every pair

of 2 incident edges $e, f$ into the clause $(e \vee f)$. It can be seen that every satisfying assignment corresponds to a matching and vice-versa.

Therefore it is $\#P$-hard. (See [3]) Note: The number of solutions of an instance Positive #2SAT can be shown to be equal to the number of vertex covers in the graph created by transforming each variable into a vertex and each clause $(u \vee v)$ into the edge $\{u, v\}$. This is also equal to the number of cliques in the complement graph.

### 5.8 #Minimal Vertex Covers

This is equal to #maximal cliques in the complement graph. It is also equal to #minimal truth settings for positive 2-SAT (by the reduction described previously).

The parsimonious reduction from bipartite #maximal matchings to #minimal truth settings for positive 2SAT is the same as above. Here there is an exact correspondence between minimal satisfying assignments ($|E|-i$ true variables) and maximal matchings (matching of size $i$). Therefore #Minimal 2SAT and #Minimal vertex cover are $\#P$-hard. (See [3])

### 5.9 Some more problems

3 Regular Bipartite Planar #Vertex Covers is equivalent to planar positive 2SAT-3 (where each clause contains variables corresponding to vertices of different colors). It is $\#P$-complete.

In addition (2-3) Regular Bipartite #Perfect Matchings is also $\#P$-complete.

Note: The decision version of these problem is easy.

## 6 Another Solution Problems (ASP)

Given an $NP$ search problem A.
ASP A: Given one solution for A, is there another?

This is useful in puzzle design, we want to be able to show that there is a unique solution.

### 6.1 Examples

- ASP $k$-coloring is trivially in $P$ since we can permute the colors.
- ASP 3-regular Hamiltonian cycle is in $P$ since it has been proved that there is always another solution.

### 6.2 ASP reduction

A problem $A$ is ASP reducible to $B$ if there is a parsimonious reduction from $A$ to $B$ with the additional property that given a solution for an instance of $A$, a solution for an instance of $B$ can be computed in polynomial time.

$A$ is ASP reducible to $B$ implies the following:

- ASP $A$ is reducible to ASP $B$ using an $NP$ reduction.
- ASP B $\in P \implies$ ASP A $\in P$.
- ASP A is $NP$-hard $\implies$ ASP B $NP$-hard.

### 6.3 ASP-Hard and ASP-Complete

ASP-Hard is defined as the set of problems ASP reducible from every $NP$ search problem. ASP-Hard $\implies$ NP-Hard.

ASP-Complete is the set of ASP-Hard $NP$ search problems. This includes all problems A for which we proved #A is $\#P$-complete.

Slitherlink was proved to be ASP-Complete.

## References


[1] Erik D. Demaine, Yoshio Okamoto, Ryuhei Uehara, and Yushi Uno. Computational complexity and an integer programming model of shakashaka. *IEICE Transactions*, 97-A(6):1213–1219, 2014.

[2] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[3] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[4] Leslie G. Valiant. Negative results on counting. In *Theoretical Computer Science, 4th GI-Conference, Aachen, Germany, March 26-28, 1979, Proceedings*, pages 38–46, 1979.