#CLIQ is #P-Complete
An Exposition by Lance Fortnow and William Gasarch

1 Introduction

Def 1.1

1. Let $B \in$ NP. Let #B be the function that, on input x, outputs the number of witnesses for $x \in A$.

2. Let $B \in$ NP. #B is #P-complete if, for all $A \in$ NP #A $\leq_T^p$ #B (you can compute #A with a polynomial Oracle Turing machine that makes queries to #B. From the proof of the Cook-Levin theorem one can see that #CNFSAT (henceforth #SAT) is #P-complete.

3. $A \leq_m^p B$ by a *parsimonious reduction* (henceforth *pars*) if there is a reduction that preserves the number of witnesses. Clearly if $A \leq_m^p B$ by a pars reduction then #A $\leq_T^p$ #B. We don't know any examples of NP-complete problems whose counting versions are not #P-complete under parsimonious reductions.

4. CLIQ is the set $\{(G, k) :$ Graph G has a clique of size $k\}$

The following are true.

1. Valiant [2, 3] defined #P and #P-complete to show that computing the permanent of a matrix is hard. He did indeed show that PERM is #P-complete (PERM can be phrased as an NP-counting problem). He also showed this for 0-1 matrices.

2. Valiant also showed several other problems #P-complete.

3. Toda [1] showed that every set in the Poly Hierarchy is poly-Turing reducible to #SAT.

#CLIQ is #P-complete. The standard proof of the Cook-Levin theorem gives a parsimonious reduction the number of witnesses of an NP algorithms to CNF-SAT. The first proof we give gives a parsimonious reduction from CNF-SAT to CLIQ. We believe this proof or something similar has long been known but we have not found it in the literature or on-line. The second proof is probably less well known and is not parsimonious.

2 Easy Proof that #CLIQ is #P-complete

Theorem 2.1

1. *There is a pars reduction from* #SAT *to* #3SAT.
2. *There is a pars reduction from* #3SAT *to* #CLIQ.

Proof:
1) A pars reduction from #SAT to #3SAT. The usual reductions are not parsimonious. We make them parsimonius by either forcing each additional variable to false, or to be forced from the assignment to the original variables.

1. Input $\phi = C_1 \wedge \cdots \wedge C_k$ where each C_i is an OR of literals.
2. Introduce three new variables x, y, z and the clauses

   $(x \wedge y \wedge \neg z) \wedge (x \wedge \neg y \wedge z) \wedge (x \wedge \neg y \wedge \neg z) \wedge$

   $(\neg x \wedge y \wedge z) \wedge (\neg x \wedge y \wedge \neg z) \wedge (\neg x \wedge \neg y \wedge \neg z) \wedge (\neg x \wedge \neg y \wedge \neg z)$

   Hence in any satisfying assignment x, y, z are all set to false.
3. For all C_i that are 1-clauses, $C_i = w$, replace it with $w \vee x \vee y$.
4. For all C_i that are 2-clauses, $C_i = w_1 \vee w_2$, replace it with $w_1 \vee w_2 \vee x$.
5. For all C_i that are 3-clauses, no change needed.
6. We now talk about C_i that have ≥ 4 literals. We do an example of what to do with a 5-clause. Note that all the $L's$ are literals, so they can be variables or their negations.

   Given $L_1 \vee L_2 \vee L_3 \vee L_4 \vee L_5$

   Introduce a new variables W. Consider

   $$(L_1 \vee L_2 \vee W) \wedge (\neg L_1 \vee L_2 \vee W) \wedge (L_1 \vee \neg L_2 \vee W) \wedge (\neg L_1 \vee \neg L_2 \vee \neg W) \wedge$$

   $$(\neg W \vee L_3 \vee L_4 \vee L_5)$$

   If $W = F$ then we get all solutions where $L_1 = T$ or $L_2 = T$.

If $W = T$ then we get all solutions where $L_1 = F$ and $L_2 = F$ but $L_3 = T$ or $L_4 = T$ or $L_4 = T$.

Hence the number of satisfying assignments is the same. But we still have a 4-clause.

Do the same procedure on the 4-clause that we did on the 5-clause.

A pars reduction from #3SAT to #CLIQ.

1. Input $\phi = C_1 \wedge \cdots \wedge C_k$ where each C_i is a 3-clause. We assume that every variable occurs in at least one clause in ϕ.
2. We create a graph G with $7k$ vertices as follows: For each clause we have 7 vertices. Label them with the 7 ways to set the 3 vars to make the clause satisfiable. For example, for the clause $x \vee y \vee \neg z$, we have 7 vertices
   - (x=T,y=T,z=T)
   - (x=T,y=T,z=F)
   - (x=T,y=F,z=T)
   - (x=T,y=F,z=F)
   - (x=F,y=T,z=T)
   - (x=F,y=T,z=F)
   - (x=F,y=F,z=F)
3. Only put an edge between vertices that are consistent with each other, i.e., don't set the same variable different ways. Note there will be no edges between two vertices assigned to the same clause.

We leave the reader to show that the number of k-cliques in G is the same as the number of satisfying assignment to ϕ. ∎

3 A Hard Non-parsimonious Proof that #CLIQ is #P-Complete

1. Valiant [2] showed that PERM (the permanent of a matrix) is #P-complete, even when restricted to 0-1 matrices.

2. View an $n \times n$ 0-1 matrix as a bipartite graph G with n vertices on both sides. The PERM is the number of perfect matchings of G. Let PM be the problem of, given a bipartite graph with n vertices on each side, does it have a perfect match. Hence #PM is #P-complete. This is interesting since PM itself is in P.

3. (Valiant proved this) Let TH-POS-2SAT be the set of all (ϕ, k) such that ϕ is a 2CNF Boolean formula with all literals positive that has a satisfying assignment with $\geq k$ of the variables set to F.

   $PM \leq$ TH-POS-2SAT with a pars reduction:

   (a) Input G, a bipartite graph with k vertices on each side.

   (b) Form the following 2CNF formula ϕ: for each edge (x, y) in G we have the variable v_{xy}. Two variables are in a clause iff the associated edges are incident.

   (c) Output (ϕ, k).

   We have a bijection from the perfect matchings of G to the satisfying assignments that have $\geq k$ variables set to F.

   If $e_1, \ldots, e_k$ is a perfect matching then the variables $e_1, \ldots, e_k$ can all be set to F (and set the rest to T) without any clause being false.

   Hence #TH-POS-2SAT is #P-complete.

4. There is a pars reduction TH-POS-2SAT $\leq$ CLIQ.

   (a) Input (ϕ, k), a 2CNF formula with no negated vars.

   (b) Form the following graph G for each vertex x we have a node x. Two variables are in a clause iff they are NOT connected by an edge.

   (c) Output (G, k).

We have a bijection from the satisfying assignments of ϕ that have k variables set to F to the cliques with k vertices. If $x_1, \ldots, x_k$ are the variables that are set to F then make those the vertices. They form a clique since no two of them can be in a clause, so no two can be an edge.

Hence #CLIQ is #P-complete.

This is a much harder proof that #CLIQ is #P-complete than the one in Theorem 2.1 since this proof uses that PERM is #P-complete, which is a difficult result.

References

[1] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal of Computing*, 20(5):865–877, 1991.
`https://doi.org/10.1137/0220053`.

[2] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
`http://dx.doi.org/10.1016/0304-3975(79)90044-6`.

[3] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):410–421, 1979.
`http://dx.doi.org/10.1137/0208032`.