CSE200 Lecture Notes – Coping with NP-completeness

Lecture by Russell Impagliazzo
Notes by Jiawei Gao

February 16, 2016

1 Coping with NP-completeness (and other intractability)

Say we have an optimization problem. To put the problem in P, an algorithm must:

- On *every* input, the running time must be a fixed polynomial.
- On *every* input, it must return the optimal solution for that input.

If an algorithm is not in P, then we have the negation of the above:

- Either on *some* inputs, the running time is not polynomial,
- Or on *some* inputs, the solution is exactly optimal.

If P_1 is polynomial-time reducible to P_2, it means hard instances of P_1 can be reduced to problem P_2, easy instances of P_1 can be reduced to problem P_2, and anything in the middle can also be reduced to problem P_2. Statement "P_2 is NP-complete" means P_2 has hard instances, but does not mean every instances of P_2 is hard.

From a real-life problem (described in real-life terms) we want to solve, we often formalize the problem, specify what are the instances, solution and objective.

What we are doing is essentially a *reduction* from real-life problem to the formalized problem. If the formalized problem turns out to be easy, then it's good. On the other hand, if the formalized problem is proved to be hard, then it does not mean the real-life problems are hard – perhaps the hard instances are not instances of the real-life problem.

Things I can do if my problem turns out to be NP-complete:

(A) **Do I really need to solve *that* problem?** Sometimes we just need to solve a restricted version of the problem that includes real-world instances.

Example 1: max independent set for interval graphs $\leq$ max independent set for general graphs. The former is in P and the latter is NP-complete.
Example 2: max independent unit disks $\leq$ max independent set for general graphs. This time the left hand side is NP-complete.

(B) **Is exponential-time algorithms OK for my instances?** Exponential-time does not mean they are slow. It means they don't scale well. If the instances are small in size, then exp-time is acceptable. Possible techniques we can use include:

1. **Improved exponential-time algorithms** (exponential, but already faster than exhaustive search.)
   Example 1: Best known algorithm for independent set is about $O(2^{0.25n})$.
   Example 2: Unit disk independent set of size k takes time $O(n^{\sqrt{k}})$.
2. **Multi-parameter analysis of algorithms** (or Fixed-parameter tractability, if the parameters are very small)
   If the problem is exponentially hard in small parameters, it's better than exponential in large parameters.
3. **Exponential-time in worst case, but not always**
   Examples: SAT-solver and algorithm for integer linear programming

(C) **Is non-optimality OK?**
A fit person and a non-fit person are being chased by a bear. The fit person says:"It's worth it after spending so much time in the gym." The non-fit person says:"Why? you won't outrun the bear." The fit person says:"I don't need to outrun the bear. I just need to outrun you."
It's OK if our algorithm just outrun other algorithms.

1. **Approximation schema**: $2^{1/\epsilon}\text{poly}(n)$ time, $(1+\epsilon)$-approximation.
   Example: Knapsack problem.
2. **Fixed constant approximation**: $(c \cdot OPT)$ solutions, where c is constant.
3. **Just heuristic performance.** (Other algorithms are even worse.)
   Randomized algorithms are in this category.

(D) **Average-case analysis**
Example: Quicksort (non-randomized) runs worst when the input is sorted or nearly sorted. But sorted or nearly sorted input is common in real-world.

2 Polynomial Hierarchy

Circuit Minimization Problem

- Input: boolean circuit C on input $x_1, \ldots, x_n$.
- Solution: boolean circuitC' on input $x_1, \ldots, x_n$.
- Constraint: For all $x_1, \ldots, x_n$, $C(x_1, \ldots, x_n) = C'(x_1, \ldots, x_n)$.
- Objective: minimize $|C'|$.

Circuit Minimization is an optimization problem, but it is not in OptP.

If P = NP then Circuit Minimization ∈ P

Negating the constraint, we get $\exists x, C'(x) \oplus C(x) = 1$. By treating $C'(x) \oplus C(x)$ as a large circuit, the negation of the constraint is a Circuit SAT problem. Using the previous conclusion "Circuit SAT ∈ NP", if P = NP, Circuit SAT ∈ P. Because P is closed under complement, ¬Circuit SAT ∈ P. Thus Circuit Min ∈ OptP = NP = P.

Definition 2.1. Let L be a language.

- P^L is the class of problems poly-time Turing reducible to L.
- NP^L is the class of problems with witnesses verifiable in P^L.

Definition 2.2. For a class of problem $\mathscr{C}$, $\mathsf{P}^{\mathscr{C}} = \bigcup_{L \in \mathscr{C}} \mathsf{P}^L$.

Because SAT is complete in NP, we get $\mathsf{P}^{\mathsf{NP}} = \mathsf{P}^{\mathrm{SAT}}$.

Proposition 2.1. If $\mathscr{C}_1 = \mathscr{C}_2$, then for any class $\mathscr{C}_3$, $\mathscr{C}_3^{\mathscr{C}_1} = \mathscr{C}_3^{\mathscr{C}_2}$. But it might not be true that for any class $\mathscr{C}_3$, $\mathscr{C}_1^{\mathscr{C}_3} = \mathscr{C}_2^{\mathscr{C}_3}$.

Next we define classes Σ_i^{P} by induction.

Definition 2.3.

- $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$.
- $\Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_1^{\mathsf{P}}}$.

Proposition 2.2. Circuit Min $\in \mathsf{P}^{\mathsf{NP}^{\mathsf{NP}}}$.