Khoury College of Computer Sciences
Northeastern University
CS 7870: Seminar in Theoretical Computer Science

Fall 2023
31 October 2023
Scribe: Omer Wasim

Outline:

- Psuedo-polynomial time and smoothed analysis
- Equivalence for binary optimization problems

1 Introduction

In the previous lecture, we introduced smoothed analysis and saw an example of its utility in establishing a polynomial time guarantee on the expected running time of a popular local search heuristic for TSP. In this example, we go deeper and look at a stronger connection of smoothed complexity with pseudo polynomial time algorithms. We study smoothed analysis in the context of binary optimization and present an amazing result that shows that whenever a smoothed polynomial time algorithm exists, we can obtain a pseudo polynomial time algorithm and vice versa! Thus, for a wide variety of problems that can be cast as binary optimization instances, the smoothed complexity class and pseudo polynomial time complexity classes are equivalent. The equivalence holds more generally, i.e. beyond binary optimization. In this lecture, we focus attention on the the equivalence for binary optimization problems. The equivalence result is due to Beier and Vöcking [BV06], and our presentation is based on the excellent lecture notes of Roughgarden [Rou17].

2 Preliminaries

Recall that the smoothed complexity of algorithm A is defined as,

$$S(n) = \sup_{x \text{ s.t. } |x|=n} \mathbb{E}_{r(\sigma)}[c(A, x + r(\sigma))]$$

where $c(A, x + r(\sigma))$ is the running time of algorithm of A on instance $x + r(\sigma)$ where $r(\sigma)$ is the perturbation to x according to some distribution. A problem is said to admit a smoothed polynomial time algorithm, if there exists an algorithm with smoothed complexity which is polynomial in the input size (i.e. n) and $\frac{1}{\sigma}$. As before, we assume that the perturbation is not spiky, and the probability density function $f(x)$ of the perturbation function satisfies $f(x) < \frac{1}{\sigma}$.

Consider a binary optimization problem in canonical form:

$$\begin{aligned}
&\max \sum_{i=1}^{n} v_i x_i \\
&x \in F \quad \text{where } F \text{ is a feasible set} \\
&x \in \{0,1\}^n, v_i \in \mathbb{Z}^+ \ \forall i \in [n]
\end{aligned}$$

Some well-studied problems in combinatorial optimization that can be cast as binary optimization problems include knapsack, subset-sum and certain variants of bin packing. For example, the knapsack problem asks to maximize the value of the knapsack where the feasible set includes all possible choices of $x \in \{0, 1\}$ where $\sum_{i=1}^{n} x_i \leq W$ where W denotes the maximum weight the knapsack can hold and v_i denotes the value of one unit of item i.

Definition 1. An algorithm for a binary optimization problem is said to take pseudo-polynomial time if it takes $O(\text{poly}(n, v_{\max}))$ time, i.e. polynomial in n and v_{max} where $v_{\max} = \max_{i \in [n]} v_i$.

Many binary optimization problems such as knapsack can be solved in pseudo polynomial time, using popular algorithmic approaches such as dynamic programming. Note that, $O(\text{poly}(v_{max}))$ is not polynomial in the input length, where the length of the input is $O(\log(\max\{n, v_{\max}\}))$.

3 Towards Equivalence

We now demonstrate the equivalence between the class of problems which admit algorithms with smoothed polynomial time algorithms and the class of problems which have pseudo-polynomial time algorithms. We attempt to show this equivalence for binary optimization problems, although this equivalence holds generally. We give a proof for the easier direction first.

3.1 Polynomial Smoothed Complexity $\implies$ Pseudo-Polynomial Complexity

The crux of the argument exploits the gap between the value of the best solution for a binary optimization problem and the value of the second-best solution. Such an argument establishing a separation between an optimal solution and other solutions is referred to as an "isolation lemma".

Let $V^{(1)}, V^{(2)}$ denote the values of the best and the second-best solutions respectively. Without loss of generality, we assume $v_i \in [0, 1] \, \forall i \in [n]$ which can be obtained by scaling all values by $\frac{1}{v_{\max}}$. A key observation is that $V^{(1)} - V^{(2)} \geq \frac{1}{v_{\max}}$. To see this, note that the *unscaled* difference between the best and second best solution is at least 1 since $v_i \in \mathbb{Z}^+$ for all v_i.

Given an algorithm $\mathcal{A}$ with smoothed polynomial time complexity, we are required to give a pseudo-polynomial time algorithm $\mathcal{A}'$ for binary optimization. The algorithm $\mathcal{A}'$ is simple: it invokes the smoothed polynomial time algorithm on a perturbed instance. Since our perturbation is small, it ensures that the solution produced by the algorithm is indeed optimal.

<u>**Algorithm $\mathcal{A}'$:**</u>

1. Perturb the values $v_i \; \forall i \in [n]$ according to a uniform distribution $U(0, \frac{1}{nv_{\max}})$, i.e. $v_i \leftarrow v_i + r_i(\sigma)$ where $r_i \sim U(0, \frac{1}{nv_{\max}}), \sigma = \frac{1}{nv_{\max}}$.
2. Run $\mathcal{A}$ on the perturbed input instance.
3. Return the solution obtained by $\mathcal{A}$.

Let us first analyze the running time of $\mathcal{A}'$ which clearly can be expressed in terms of the running time of $\mathcal{A}$ on the perturbed instance. Since $\sigma = \frac{1}{nv_{\max}}$, the running time of $\mathcal{A}$ is $O(\text{poly}(n, nv_{\max})) = O(\text{poly}(n, v_{\max}))$. Thus, $\mathcal{A}'$ is a pseudo-polynomial time algorithm.

What about the cost of the solution returned by $\mathcal{A}'$? We claim that the optimal solution to the original unperturbed problem instance is preserved under perturbation. To see this, consider the quantity $V^{(1)} - V^{(2)}$ in the perturbed instance. This is clearly bounded by $n\frac{1}{nv_{\max}} = \frac{1}{v_{\max}}$. Since $V^{(1)} - V^{(2)} \geq \frac{1}{v_{\max}}$ in the unperturbed instance, the optimal solution in the original instance stays optimal under the perturbed instance.

3.2 Pseudo-Polynomial Complexity $\implies$ Polynomial Smoothed Complexity

To prove the other direction, given a pseudo polynomial time algorithm $\mathcal{A}$ taking time $O(\text{poly}(n, v_{\max}))$, we show how to obtain a smoothed poly-time algorithm $\mathcal{A}'$ taking time $O(\text{poly}(n, \frac{1}{\sigma}))$ for binary optimization with probability at least $1 - \frac{1}{n}$.

Analysis of 'winner gap' in the perturbed instance. The perturbation distribution is the uniform distribution $U(0, \sigma)$. We upper bound the probability of the best and second best solutions differing by at most ϵ for some $\epsilon > 0$. To this end, fix some i. We say that index i is ϵ-bad if the optimal solution with i included is within ϵ of the optimal solution with i excluded. Let S_{+i} denote the value of the best solution solution with i included–i.e. $x_i \neq 0$ and S_{-i} denote the value of the best solution with i excluded. Let S_{+i-i} denote the quantity $S_{+i} - v_i$. Fixing the values of all v_j where $j \neq i$, and using the principle of deferred decisions, we want to analyze $\Pr[i \text{ is } \epsilon - \text{bad}]$. By the perturbation assumption, this happens if $S_{+i-i} + v_i + r_i(\sigma) \in [S_{-i} - \epsilon, S_{-i} + \epsilon]$. The probability of this happening is bounded by $\Pr[r_i(\sigma) \leq 2\epsilon] \leq \frac{2\epsilon}{\sigma}$. This yields the following lemma.

Lemma 1. $\Pr[V^{(1)} - V^{(2)} < \epsilon] \leq \frac{2n\epsilon}{\sigma}$

Proof. From the above discussion, the probability that any index i is ϵ-bad is given by $\frac{2\epsilon}{\sigma}$. If $V^{(1)} - V^{(2)} < \epsilon$, then this implies that some index i must be ϵ-bad. Taking a union bound over the n indices upper bounds the probability of this event, which is $\frac{2n\epsilon}{\sigma}$. □

We now give the smoothed polynomial time algorithm $\mathcal{A}'$. Thereafter, we refer to v_i as the perturbed values. Assume that v_i for all $i \in [n]$ is given in the bit representation. The algorithm essentially utilizes $\mathcal{A}$ by iteratively examining the first b bits for $b = 1, 2, 3, ...b_p$ where b_p is the bit precision of the representation, runs $\mathcal{A}$ on the truncated values and checks if the solution returned by $\mathcal{A}$ is optimal for any possible setting of the remaining bits. If yes, it returns the solution returned by $\mathcal{A}$, otherwise considers one additional bit and repeats the process.

Algorithm $\mathcal{A}'$:

- For $b = 1, 2, 3, ..., b_p$:
  - Set $v_i^b = v_i$ up to b bits of precision for all $i \in [n]$.
  - Run $\mathcal{A}$ on instance $\{v_i^b\}_{i=1}^n$ to find the optimal solution x_b.

– Check if x_b is optimal for all possible choices of the remaining bits. If yes, return x_b, else continue.

We explain how $\mathcal{A}'$ implements the third step to check optimality of x_b. This is simple. Let S denote the set of indices for which $x_i \neq 0$, and $\bar{S}$ denote the remaining indices. Let $v_i^{b*} = v_i^b + \frac{1}{2^b}$ for all $i \notin S$ and $v_i^{b*} = v_i$ for $i \in S$. We run $\mathcal{A}$ on the input $\{v_i^{b*}\}_{i=1}^n$ and check if the optimal solution remains the same. If yes, the algorithm returns x_b, otherwise continues to another iteration of the for loop.

Clearly, the running time of $\mathcal{A}'$ depends on the number of iterations or bits we examine before converging to an optimal solution. We prove the following theorem, thereby establishing that $\mathcal{A}'$ has smoothed complexity $O(\text{poly}(n, \frac{1}{\sigma}))$ with probability at least $1 - \frac{1}{n}$.

Theorem 1. With probability at least $1 - \frac{1}{n}$, $\mathcal{A}'$ terminates after $b = \Theta(\log(\frac{n}{\sigma}))$ iterations.

Proof. By Lemma 1, we know that $\Pr[V^{(1)} - V^{(2)} < \epsilon] \leq \frac{2n\epsilon}{\sigma}$. Consider iteration b. If $|v_i^b - v_i| \leq \frac{1}{2^b}$, then for any solution x, the value of the solution on the truncated values on b bits, $\{v_i^b\}_{i=1}^n$ differs by at most $\frac{n}{2^b}$. Thus, if the winner gap $|V^{(1)} - V^{(2)}| \geq \frac{n}{2^b}$, then the optimal solution for the instance truncated to b bits is also optimal for the untruncated instance. We want the failure probability to be at most $\frac{1}{n}$, so letting $\frac{2n\epsilon}{\sigma} < \frac{1}{n}$ yields $\epsilon < \frac{\sigma}{2n^2}$, and requiring $\epsilon > \frac{n}{2^b}$ to preserve optimality implies $\frac{n}{2^b} < \frac{\sigma}{2n^2}$. This yields $2^b \geq \frac{2n^3}{\sigma}$ so that setting $b = \Theta(\log(\frac{n}{\sigma}))$ completes the proof. □

The total running time of $\mathcal{A}$ is bounded within a b factor of the running time of $\mathcal{A}$. The running time of $\mathcal{A}$ is $O(\text{poly}(n, v_{\max}))$, and since v_{max} can be represented by a value at most $2^b = \Theta(\frac{n}{\sigma})$, we obtain that the running time of $\mathcal{A}'$ is at most $b \cdot \Theta(\frac{n}{\sigma}) = O(\text{poly}(n, \frac{1}{\sigma}))$ with probability at least $1 - 1/n$.

We have thus shown that if there is a pseudo-polynomial time algorithm for a binary optimization algorithm, then there exists an algorithm that for any instance x yields an optimal solution for a perturbation of x in time $O(\text{poly}(n, \frac{1}{\sigma})$ with probability at least $1 - 1/n$. This can be generalized to show that for any $\delta > 0$, the algorithm takes time $O(\text{poly}(n, \frac{1}{\sigma}, \frac{1}{1/\delta})$ with probability at least $1 - 1/\delta$. Technically, this is not the same as saying that there is a smoothed polynomial time algorithm since we have not bounded the expected time for completion. We refer the reader to [BV06] for the stronger result.

References

[BV06] Rene Beier and Berthold Vöcking. Typical properties of winners and losers [0.2ex] in discrete optimization. *SIAM Journal on Computing*, 35(4):855–881, 2006.

[Rou17] T. Roughgarden. Smoothed complexity and pseudo-polynomial time algorithms. Lecture notes, available from `https://timroughgarden.org/w17/l/l18.pdf`, 2017.