Approximation Algorithms

Oran Danon

§1. Introduction

§ 1.1 Definitions.

Let μ denote a maximization problem, and let ν denote a minimization problem.

A poly-time algorithm $\mathcal{A}$ satisfying:

$$|\mathcal{A}(I)|^{a} \geq C \cdot \mathbf{OPT}(I)$$

for all instance I of μ, is said to be an **approximation algorithm for** μ with **approximation ratio** C.

[a] $|\mathcal{A}(I)|$ denotes what the algorithm $\mathcal{A}$ output on instance I

Remark 1. 1-approximation algorithm is one that solves the problem optimally. Hence we would like to get approximation ratio as close to one as possible.

Definition 2. *A sequence of algorithms $\mathcal{A}$ is said to be an* ***approximation scheme*** *for μ if for every instance I of μ and for every $\varepsilon > 0$:*

$$|\mathcal{A}(I, \varepsilon)| \geq (1 - \varepsilon)\, \mathbf{OPT}(I).$$

Moreover, such a scheme is said to be a ***poly-time approximation scheme*** *and abbreviate* ***PTAS*** *if for every $\varepsilon > 0$ the running time of $\mathcal{A}$ is bounded by* $\mathbf{poly}(|I|)$.

Remark 3.

- Here ε is part of the input.
- Why do we say scheme?
  - Because we can treate $\mathcal{A}$ as a family of algorithms i.e. $\{\mathcal{A}_\varepsilon\}_{\varepsilon > 0}$.
- As an example for PTAS, take $O(n^{2/\varepsilon})$, then for every ε this is a polynomial.

Definition 4. *A PTAS whose running time is bounded by a polynomial of the size of the instance and $1/\varepsilon$ is called a* ***fully poly-time approximation scheme*** *and abbreviate* ***FPTAS****.*

Remark 5. As an example one can take $O\left((1/\varepsilon)^2 n^3\right)$

> For NPC problems, an FPTAS is the best one can hope for.

§ 1.2 Knapsack Problem.

Instance: items $a_1, a_2, \ldots, a_n$ with weights $w(a_i)$ and profits $p(a_i)$ for all $1 \le i \le n$, and a total weight $W \in \mathbb{R}$.
Goal: a boolean vector $(x_1, x_2, \ldots, x_n)$ such that

$$\max \sum_{i=1}^{n} x_i p(x_i) \text{ such that } \sum_{i=1}^{n} x_i w(a_i) \le W$$

For knapsack problem one can obtain $O(nW)$ algorithm using dynamic programming as follows:

1. For $i \in \{1, 2 \ldots, n\}$ and $0 \le j \le W$ set $S(i,j)$ to be the optimal if we can only choose from $\{a_1, \ldots, a_i\}$ and the sack limitation is j.

2. Then set

$$S(i,j) = \begin{cases} \max\{S(i-1,j), p(a_i) + S(i-1, j - w(a_i))\} & , \ w(a_i) \le j \\ S(i-1,j) & , \ otherwise \end{cases}$$

The running time of this algorithm is pseudo-polynomial.

Definition 6. *If I is an instance of μ we write $|I_n|$ to denote the length of the unary encoding of I.*

Definition 7. *An algorithm A for μ is said to be pseudo-polynomial if its running time is bounded by* $\mathbf{poly}(|I_n|)$ *for all instance I of μ.*

Remark 8.

- Recall that time complexity $\ge$ space complexity demonstrates why unary Unicode is unreasonable.

- For knapsack: The algorithm above runs in $\Theta(nW)$ knapsack has a pseudo-poly algorithm.

types of problems.pdf

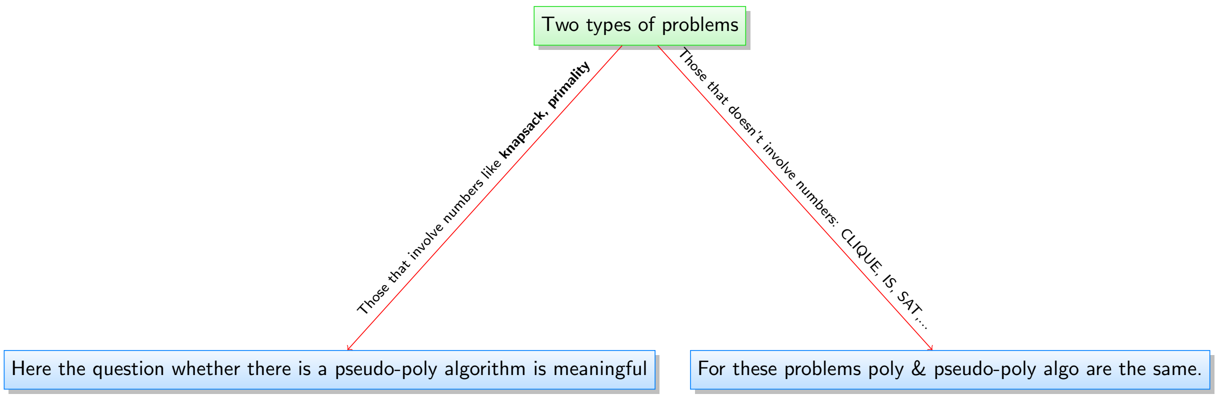

Figure 1: Two types of problems

Thus we can conclude that being NPC does not exclude not having a pseudo-poly algorithm.

Definition 9. *Given a decision problem μ and an instance I of μ define:*

- $\mathrm{Max}(I) :=$ *The largest integer present in I.*
- $\mathrm{Length}(I) :=$ *length of the encoding of I under a reasonable encoding scheme (say binary).*

Definition 10. *Problem μ is said to be a* ***number problem*** *if there exists no poly $p(\cdot)$ such that*

$$\mathrm{Max}(I) \leq p(\mathrm{Length}(I))$$

Definition 11. *An algorithm is said to be pseudo-polynomial if the algorithm is polynomial in* $\mathrm{Length}(I)$ *and* $\mathrm{Max}(I)$ *for every I.*