Algorithm Design, Analysis & Complexity

Lecture 11 - Approximation Algorithms

Koushik Pal

University of Toronto

August 3, 2021

$\mathcal{NP}$-completeness - one more example

Theorem

TravelingSalesman *is $\mathcal{NP}$-Complete.*

Proof.

It is easy to see that TravelingSalesman is in $\mathcal{NP}$: the certificate is a permutation of the cities and a certifier checks that the length of the corresponding tour is at most the given bound.

We will now show that HamCycle $\leq_p$ TravelingSalesman: given an undirected graph $G = (V, E)$ with $|G| = n$, we define the following instance G' of TravelingSalesman: we have a city v_i' for each node v_i of the graph G, and we set

$$
\begin{aligned}
d(v_i', v_j') &:= \begin{cases} 1 & \text{if there is an edge } (v_i, v_j) \in G, \\ 2 & \text{otherwise.} \end{cases} \\
k &:= n
\end{aligned}
$$

It should be easy to verify that this transformation can be done in polytime.

Proof

We claim that G has a Hamiltonian cycle iff there is a tour of length at most n in G'.

For, if G has a Hamiltonian cycle, then this ordering of the corresponding cities defines a tour of length n.

Conversely, suppose there is a tour of length at most n. The expression for the length of this tour is a sum of n terms, each of which is at least 1. Thus, it must be the case that all the terms are equal to 1. It follows that the ordering of these corresponding nodes must form a Hamiltonian cycle in G. $\square$

Self Reducibility

Search Problems: Given input x, find solution y.

Clearly, efficient solution to search problems would give efficient solution to corresponding decision problems. For example,

```
1: procedure IndependentSet-Decision(G, k)
2:     return IndependentSet-Search(G, k) != Nil
```

Definition

A search problem is **self-reducible** if any efficient solution to the decision problem can be used to solve the search problem efficiently.

Examples

Example

- HamPath-Search

procedure HamPath-Search(G)
  if not HamPath-Decision(G) **then**
    return Nil
  for each edge $e \in E$ **do**
    $E' = E \setminus \{e\}$
    $G' = (V, E')$
    if HamPath-Decision(G') **then**
      $E = E'$
  return E

Examples

Correctness: $\textsc{HamPath-Decision}(G)$ remains true at every step. So, at the end, E contains every edge in a Hamiltonian path of G. At the same time, every other edge is taken out because it is not required. So, E contains no other edge. Hence, the value returned is a Hamiltonian path of G.

Runtime: Let $t(n, m)$ denote the runtime of $\textsc{HamPath-Decision}$ on a graph of n vertices and m edges.

Then, total runtime $= \mathcal{O}((m + 1) \times t(n, m) + m)$.
This is polytime if $t(n, m)$ is.

Examples

Example

- VertexCover-Search

procedure VertexCover-Search(G, k)
  if not VertexCover-Decision(G, k) **then**
    return Nil
  $C = \{\}$
  for each vertex $v \in V$ and while $k > 0$ **do**
    if VertexCover-Decision$(G \setminus \{v\}, k-1)$ **then**
      $C = C \cup \{v\}$
      $V = V \setminus \{v\}$
      $E = E \setminus \{(v, w) \mid w \in V\}$
      $G = (V, E)$
      $k = k-1$
  return C

Examples

Correctness: If $G \setminus \{v\}$ contains a vertex cover of size $k-1$, say C', then $C' \cup \{v\}$ is a vertex cover of size k in G.

Conversely, if $G \setminus \{v\}$ does not contain a vertex cover of size $k-1$, then v does not belong to any vertex cover of size k in G.

Runtime: Let $t(n, m)$ denote the runtime of VertexCover-Decision on a graph of n vertices and m edges.

Then, total runtime $= \mathcal{O}((n+1) \times t(n, m) + n \times (n+m))$.
This is polytime if $t(n, m)$ is.

Optimization Problems

Optimization Problems: Given input x, find solution y that minimizes or maximizes a given function; e.g., given G, find an independent set of the maximum size.

This optimization problem is **self-reducible**: Do binary search in range $[1, n]$ by making calls to $\textsc{IndependentSet-Decision}$ (G, k) for various values of k in order to find value k such that G contains a k-independent set but not a $(k+1)$-independent set.

Similar idea for minimum vertex cover.

How to handle intractable problems?

Traditional point of view

- $\mathcal{P}$ = "easy"
- $\mathcal{NP}$ = "hard".

But

- definition of $\mathcal{NP}$-completeness is based on worst-case analysis (maybe inputs encountered in practice are rarely worst-case)
- for real-world input sizes ($\geq 10^9$), even n^2 runtime is inefficient!

Fact

It is not possible to solve $\mathcal{NP}$-hard problems "exactly" and "in polytime" unless $\mathcal{P} = \mathcal{NP}$.

How to handle intractable problems

1. In practice, sometimes sacrifice on efficiency: use exponential time algorithm and hope that the inputs don't trigger worst-case behaviour.
2. A problem that is hard in general may be easy for a certain restricted class of inputs, e.g.,
   2.1 2-Sat
   2.2 Dnf-Sat
   2.3 2-DimensionalMatching
   2.4 2-Coloring
   2.5 IndependentSet on a Tree
3. Sometimes sacrifice on exactness, particularly for optimization problems: instead of searching for "best" solution, settle for "good enough" solution.

Approximation Algorithm

Definition

For minimization problems, let $OPT(x)$ be the minimum value of any solution for input x. Suppose we have an **approximation algorithm** that generates solutions with approximate value $A(x)$. By definition, $OPT(x) \leq A(x)$ for all x.

The **approximation ratio** of our algorithm is a function $r(n)$ such that $A(x) \leq r(n) \times OPT(x)$ for all n and all inputs x of size n.

Approximation Algorithm for VERTEXCOVER

Example

procedure VERTEXCOVER-OPTIMIZATION(G)
    $C = \emptyset$
    $E' = E$
    while $E' \neq \emptyset$ **do**
        Let (u, v) be an arbitrary edge of E'
        $C = C \cup \{u, v\}$
        $E'' = E' \setminus \Big(\{(u, w) \mid (u, w) \in E'\} \cup \{(v, w) \mid (v, w) \in E'\} \Big)$
        $E' = E''$
    return C

Approximation Algorithm for VertexCover

Claim: $|C| \leq 2 \times OPT$.

Proof.

Any vertex cover of G must include at least one endpoint from every edge in C (all edges in C are disjoint, i.e., with no endpoint in common). So, in particular, $OPT \geq \frac{|C|}{2}$. This shows that approximation ratio ≤ 2.

To show approximation ratio $= 2$, we need an example where algorithm performs that badly: use n disjoint edges! Algorithm returns $2n$ vertices, while n of them are sufficient. $\square$

Lower bound approach

Question

In general, how can we compute ratio without knowing OPT?

Answer: Use a lower bound! For another value LB that's easy to compute and for which you can prove $LB \leq OPT$, show that $A \leq r \times LB$.

Approximation Algorithm for VertexCover revisited

Create a linear program from the input graph as follows:

$$\begin{array}{ll} \text{minimize} & x_1 + \cdots + x_n \\ \text{subject to} & x_i + x_j \geq 1 \text{ for each } (v_i, v_j) \in E \\ & x_i \in \{0, 1\} \text{ for all } i = 1, \ldots, n, \end{array}$$

where x_i corresponds to node v_i for $i = 1, \ldots, n$.

This is an Integer Programming Problem (IPP). Convert it to an LPP by changing the last constraint to:

$$0 \leq x_i \leq 1 \quad \text{for all } i.$$

Now, compute an optimal solution to this LPP, say $(x_1^*, \ldots, x_n^*)$.

Approximation Algorithm for $\textsc{VertexCover}$ revisited

Create a vertex cover C of G as follows:

$$\text{for each } v_i \in V, \text{ put } v_i \in C \iff x_i^* \geq \frac{1}{2}.$$

C is a cover because the constraint $x_i + x_j \geq 1$ guarantees that at least one of $x_i^*, x_j^* \geq \frac{1}{2}$ for each edge $(v_i, v_j) \in E$. In particular, one of v_i, v_j belongs to C for each edge $(v_i, v_j) \in E$.

Question: What is the approximation ratio?
Answer: 2.

Approximation Algorithm for VertexCover revisited

Proof.

Consider a minimum vertex cover C'. For $i = 1, \ldots, n$, let

$$\begin{aligned} x'_i &= 1 \quad \text{if } v_i \in C', \\ x'_i &= 0 \quad \text{otherwise.} \end{aligned}$$

Then $\{x'_1, \ldots, x'_n\}$ is a solution to the IPP. So,

$$|C'| = \sum_{i=1}^{n} x'_i \geq \sum_{i=1}^{n} x^*_i,$$

where $\{x^*_1, \ldots, x^*_n\}$ is an optimal solution to the LPP with no restriction on values, so guaranteed to be at least as small as any other solution, including those with additional restrictions.

Thus, $LB = \sum_{i=1}^{n} x^*_i$, and it is easy to compute.

Approximation Algorithm for VERTEXCOVER revisited

For $i = 1, \ldots, n$, let

$$\begin{array}{lll} \tilde{x}_i = 1 & \text{if } v_i \in C & \text{if } x_i^* \geq \dfrac{1}{2} \\ \tilde{x}_i = 0 & \text{if } v_i \notin C & \text{if } x_i^* < \dfrac{1}{2}. \end{array}$$

Then, $|C| = \sum_{i=1}^{n} \tilde{x}_i$.

Also, $\tilde{x}_i \leq 2x_i^* = 2 \times LB$ for each i. So,

$$|C| = \sum_{i=1}^{n} \tilde{x}_i \leq 2 \sum_{i=1}^{n} x_i^* \leq 2 \sum_{i=1}^{n} x_i' \leq 2 \times |C'|.$$

Hence, C is no more than twice the size of a minimum vertex cover. $\square$

N.B.: This technique is called LP Relaxation.

How well can $\mathcal{NP}$-complete problems be approximated?

Even though all $\mathcal{NP}$-complete problems are "equivalent" to each other (in some sense), approximation ratios for corresponding optimization problems are all over the place.

- $\textsc{VertexCover}$: constant approximation ratio of 2
- $\textsc{Knapsack}$: approximation ratio of $1 - \epsilon$ with time complexity $\mathcal{O}(\frac{n^3}{\epsilon})$ for all constants $\epsilon \in (0, 1]$.
- $\textsc{TravelingSalesman}$: no constant ratio unless $\mathcal{P} = \mathcal{NP}$!