Polynomial Time Reduction

- Polynomial Time Reduction Definition
- Reduction by Equivalence
- Reduction from Special Cases to General Case
- Reduction by Encoding with Gadgets
- Transitivity of Reductions
- Decision, Search and Optimization Problem
- Self-Reducibility

Imdad ullah Khan

Versions of Problems: Decision Problems

Decision Problem

- Sometimes called decision version of a problem
- These problems can be characterized by their algorithms whose output is either **Yes** or **No**
- In other words the answer on an instance is either **Yes** or **No**
- SAT, 3-SAT are decision problems
- So are all the other problems we studied so far
- IND-SET(G, k), VERTEX-COVER(G, k), PRIME(n), CLIQUE(G, k), SET-COVER$(U, \mathcal{S}, k)$, SUBSET-SUM(U, w, C)

Versions of Problems: Search Problems

Search Problem

- Some times called search version of a problem
- These problems ask for a structure satisfying certain property or NOT-FOUND= **NF** flag
- The expected answer on an instance is not (necessarily) **Yes** or **No**
- Search versions of SAT, 3-SAT ask for a satisfying assignment
  - output is n-bit string (specifying ordered values for variables) or **NF**
- Search version of IND-SET(G, k) asks for an ind. set of size k in G
  - output is a subset of vertices or **NF**
- Search version of SET-COVER$(U, \mathcal{S}, t)$
  - output is a t-sized sub collection of $\mathcal{S}$ or **NF**

Versions of Problems: Optimization Problems

Optimization Problem

- These problems ask for a structure that satisfy certain property (feasibility) and no other feasible structure have better value
- these are search problem but searching for an optimal structure
- Optimization versions of SAT, 3-SAT ask for an assignment satisfying the most number of clauses
  - output is n-bit string (specifying ordered values for variables)
- Optimization problems IND-SET(G), CLIQUE(G) ask for largest independent set or clique in a graph G
- MIN-VERTEX-COVER(G) asks for a vertex cover of minimum size
- TSP(G) asks for a minimum cost TSP tour
- As in DP, sometimes we only need value of the optimal solution

Versions of Problems

- Decision Problem: answer is **Yes/No**
- Search Problem: answer is a feasible structure of certain value or **NF**
- Optimization Problem: answer is a feasible structure of optimal value

- Some authors only use decision problems and search problems. Search problem there actually means the optimization problem
  - This perhaps is a better notion, since if you know value of the optimal solution (which can be found through decision version of the problem), then one can use search problem (our notion) with the input value equal to the optimal value
- In some cases there is no reasonable notion of optimization version e.g. HAMILTONIAN CYCLE problem and 3-COLORING PROBLEM

Versions of Problems: Self Reducibility

Are versions of a problem polynomial time reducible to each other?

- Many search and optimization problems are only polynomially more difficult than corresponding decision problem
- Any efficient algorithm for the decision problem can be used to solve the search problem efficiently
- This is called **self-reducibility**

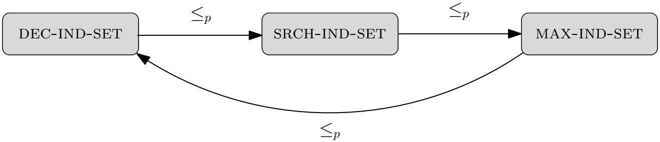
All the problems we discuss exhibit self-reducibility

Versions of Problems: Self Reducibility

Are versions of a problem polynomial time reducible to each other?

- Many search and optimization problems are only polynomially more difficult than corresponding decision problem
- Any efficient algorithm for the decision problem can be used to solve the search problem efficiently
- This is called **self-reducibility**

All the problems we discuss exhibit self-reducibility, where appropriate



By transitivity of reductions, all versions are equivalent

Versions of Problems: Self Reducibility

Are versions of a problem polynomial time reducible to each other?

$$\text{DEC-IND-SET}(G,k) \quad \leq_p \quad \text{MAX-IND-SET}(G)$$

Proof:

- Suppose $\mathcal{A}$ is an algorithm for MAX-IND-SET(G)
- Given an instance $[G,k]$ of DEC-IND-SET(G,k)
- Call $\mathcal{A}$ on G
  - if the returned independent set is of size $\geq k$, then return **Yes**
  - else return **No**
- Need to check size of the returned set (polynomial time)

Versions of Problems: Self Reducibility

$$\textsc{dec-ind-set}(G,k) \quad \leq_p \quad \textsc{srch-ind-set}(G,k)$$

Proof:

- Suppose $\mathcal{A}$ is an algorithm for $\textsc{srch-ind-set}(G,k)$
- Given an instance $[G,k]$ of $\textsc{dec-ind-set}(G,k)$
- Call $\mathcal{A}$ on $[G,k]$
  - if it returns an independent set, then return **Yes**
  - else if it returns **NF**, then return **No**

Versions of Problems: Self Reducibility

$\text{SRCH-IND-SET}(G, k) \quad \leq_p \quad \text{MAX-IND-SET}(G)$

Proof:

- Suppose $\mathcal{A}$ is an algorithm for MAX-IND-SET(G)
- Given an instance $[G, k]$ of SRCH-IND-SET(G, k)
- Call $\mathcal{A}$ on G
  - if returned independent set is of size $\geq k$, then return the set (or any k vertices out of it)
  - else return **NF**
- Need to check size of the returned set and select k of it (poly-time)

Versions of Problems: Self Reducibility

$$\text{SRCH-IND-SET}(G, k) \quad \leq_p \quad \text{DEC-IND-SET}(G, k)$$

- Let $\mathcal{A}$ be an algorithm for DEC-IND-SET(G, k). Using $\mathcal{A}$
- for each vertex we determine if it is needed for an ind. set of size k

Algorithm Algorithm for SRCH-IND-SET(G, k) problem

```
𝓘 ← ∅                                    ▷ Initialize an empty independent set
t ← k
for v ∈ V(G) do
    ans ← A(G \ {v}, t)
    if ans = yes then
        V(G) ← V(G) \ {v}
    else
        V(G) ← V(G) \ {v}
        𝓘 ← 𝓘 ∪ {v}
        t ← t − 1
```

Versions of Problems: Self Reducibility

MAX-IND-SET(G) $\leq_p$ DEC-IND-SET(G, k)

- Suppose $\mathcal{A}$ is an algorithm for DEC-IND-SET(G, k)
- First find the size of maximum independent set (optimal value)
- For $t \geq 1$, call $\mathcal{A}$ on $[G, t]$
- If it outputs **Yes** increment t until the output is **No**
- Let k be the last t for which there is a **Yes** answer
- This k is the size of max independent set
- Search for ind. set of size k using the previous algorithm
- Note that it uses monotonicity of independent sets
- Should use binary search for the last **Yes** answer, Why?
  - In some cases it may be essential to keep reduction polynomial time

Self Reducibility: Hamiltonian Path

$$\text{SRCH-HAM-PATH}(G) \quad \leq_p \quad \text{DEC-HAM-PATH}(G)$$

- Let $\mathcal{A}$ be an algorithm for DEC-HAM-PATH(G)
- Call $\mathcal{A}$ on G, if it returns **No** then return **NF**
- For each vertex v, call $\mathcal{A}$ on $G \setminus \{v\}$
- select or de-select v? All vertices have to be in Ham path

- For each edge $e = (u, v)$, call $\mathcal{A}$ on $G \setminus \{e\}$
- If it returns **Yes**, then e is not needed for Ham path, remove e from G
- If it returns **No**, then e is needed
- In the end, only edges of a Ham path will remain

Self Reducibility: Vertex Cover

$$\text{SRCH-VERTEX-COVER}(G, k) \quad \leq_p \quad \text{DEC-VERTEX-COVER}(G, k)$$

- Suppose $\mathcal{A}$ is an algorithm for DEC-VERTEX-COVER(G, k)
- Call $\mathcal{A}$ on G and k, if it returns **No**, then return **NF**
- For each vertex v, call $\mathcal{A}$ on $G \setminus \{v\}$ and k
- If G has cover of size k, then $G \setminus \{v\}$ has a VC of size k
  - whether or not v is in the cover
  - We will get **Yes** answer in both cases
- Call $\mathcal{A}$ on $G \setminus \{v\}$ and $k-1$, if it returns **Yes**, then $v \in k$-sized cover
- If it returns **No**, then v is not part of any k-sized cover

Self Reducibility: Vertex Cover

$\textsc{srch-vertex-cover}(G, k) \ \leq_p \ \textsc{dec-vertex-cover}(G, k)$

- Suppose $\mathcal{A}$ is an algorithm for $\textsc{vertex-cover}(G, k)$
- Call $\mathcal{A}$ on G and k, if it returns **No**, then return **NF**
- For each vertex v, call $\mathcal{A}$ on $G \setminus \{v\}$ and k
- $G \setminus \{v\}$ has a VC of size k whether or not v is in the cover.
- Call $\mathcal{A}$ on $G \setminus \{v\}$ and $k - 1$, if it returns **Yes**, then $v \in k$-sized cover
- If it returns **No**, then v is not part of any k-sized cover

Algorithm for $\textsc{srch-ind-set}(G, k)$ using $\mathcal{A}$ for $\textsc{dec-ind-set}(G, k)$

```
C ← ∅,          t ← k
for v ∈ V(G) = {v_1, ..., v_n} and while t ≥ 1 do
    ans ← A(G \ {v}, t − 1)
    if ans = Yes then
        C ← C ∪ {v}
        t ← t − 1
    V(G) ← V(G) \ {v}
```

Caution for Self Reducibility

CAUTION! self-reducibility **does not** mean that "any algorithm solving the decision version must use a solution of the search version"

The search version of FACTOR(n, k) problem is in a sense the *'complement of* the PRIME(n) (and COMPOSITE(n)) problem

FACTOR(n): Find a factor of n else output **NF** $\Leftrightarrow$ (n is prime)

The famous AKS (2004) theorem on primality testing uses involved number theory to **solve** the PRIME(n) and COMPOSITE(n) problem, but does not solve the search problem FACTOR(n) (no polynomial time algorithm is yet known for it)

In other words, there are search versions of the problem that are not known to be reducible to their decision versions

We focus on decision problems (or decision version of problems)