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: Self Reducibility

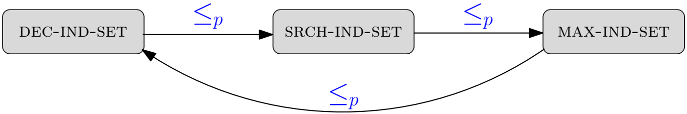
Are versions of a problem polynomial time reducible to each other?

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 ▷ w.r.t polytime

Versions of Problems: Self Reducibility

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

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

Proof: Suppose $\mathcal{A}$ is an algorithm for $\textsc{max-ind-set}(G)$

Given an instance $[G,k]$ of $\textsc{dec-ind-set}(G,k)$

1. Call $\mathcal{A}$ on G
2. if the returned independent set is of size $\geq k$, then return **Yes**
3. else return **No**
4. Need to check size of the returned set ▷ polynomial time

Versions of Problems: Self Reducibility

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

Proof: Suppose $\mathcal{A}$ is an algorithm for SRCH-IND-SET(G, k)

Given an instance $[G, k]$ of DEC-IND-SET(G, k)

1. Call $\mathcal{A}$ on $[G, k]$
2. if it returns an independent set, then return **Yes**
3. 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 $\text{MAX-IND-SET}(G)$

Given an instance $[G, k]$ of $\text{SRCH-IND-SET}(G, k)$

1. Call $\mathcal{A}$ on G
2. if returned independent set is of size $\geq k$, then return the set (or any k vertices out of it)
3. else return **NF**
4. Need to check size of the returned set and select k of it $\quad \triangleright$ 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).

We use $\mathcal{A}$ to determine if a vertex 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 is not needed
        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)

1. For $t \geq 1$, call $\mathcal{A}$ on $[G, t]$
2. If it outputs **Yes** increment t until the output is **No**
3. Let k be the last t for which there is a **Yes** answer

This k is the size of max independent set

Find a k-ind.set using previous algo (SRCH-IND-SET(G, k) $\leq_p$ DEC-IND-SET(G, k))

▷ Note that it uses monotonicity of independent sets

We should use binary search for the last **Yes** answer, Why?

▷ It may be essential to keep reduction polynomial time

Self Reducibility: Hamiltonian Path

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

Suppose $\mathcal{A}$ is an algorithm for $\text{DEC-HAM-PATH}(G)$

1. Call $\mathcal{A}$ on G, if it returns **No** then return **NF**
2. For each vertex v, call $\mathcal{A}$ on $G \setminus \{v\}$

   ▷ select or de-select v? All vertices have to be in Ham path

1. For each edge $e = (u, v)$, call $\mathcal{A}$ on $G \setminus \{e\}$
2. If it returns **Yes**, then e is not needed for Ham path, remove e from G
3. If it returns **No**, then e is needed
4. 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 $\text{DEC-VERTEX-COVER}(G, k)$

1. Call $\mathcal{A}$ on G and k, if it returns **No**, then return **NF**
2. For each vertex v, call $\mathcal{A}$ on $G \setminus \{v\}$ and k
3. If G has cover of size k, then $G \setminus \{v\}$ has a VC of size k

   $\triangleright$ whether or not v is in the cover, we will get **Yes** answer
4. Call $\mathcal{A}$ on $G \setminus \{v\}$ and $k - 1$, if it returns **Yes**, then $v \in k$-sized cover
5. If it returns **No**, then v is not part of any k-sized cover

Self Reducibility: Vertex Cover

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

Suppose $\mathcal{A}$ is an algorithm for $\textsc{dec-vertex-cover}(G,k)$

1. Call $\mathcal{A}$ on G and k, if it returns **No**, then return **NF**
2. For each vertex v, call $\mathcal{A}$ on $G \setminus \{v\}$ and k
3. If G has cover of size k, then $G \setminus \{v\}$ has a VC of size k $\triangleright$ whether or not v is in the cover, we will get **Yes** answer
4. Call $\mathcal{A}$ on $G \setminus \{v\}$ and $k-1$, if it returns **Yes**, then $v \in k$-sized cover
5. 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** $\iff$ (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)