Reductions and Approximations

CSC 349

Christopher Siu

Tractability and Solvability

Definition

A problem is **tractable** if and only if it is solvable in worst-case polynomial time.

- A problem that is not tractable is said to be **intractable**.

Example

The Sorting Problem is tractable (and, thus, solvable).

Example

The Knapsack Problem is solvable but intractable.

Example

The Halting Problem is unsolvable (and, thus, intractable).

Decision Problems

Definition

A **decision problem** is one whose solution is either "yes" or "no".

- Decision problems contrast with **optimization problems**.
  - The solution is neither "yes" nor "no".
  - The solution is associated with some "best" value.

Example

The Hamiltonian Path Problem is a decision problem.

Example

The Traveling Salesperson Problem is an optimization problem.

Decision Problems

Example

Consider the Traveling Salesperson Problem:

- Given a complete, weighted graph...
- ...find the Hamiltonian cycle of minimum weight.

- An optimization problem can be restated as a decision problem by introducing a **threshold**.

Example

Consider the decision version of TSP:

- Given a complete, weighted graph and a threshold k...
- ...find a Hamiltonian cycle of weight $\leq k$.

Complexity Classes

Definition

A **complexity class** is a set of problems with similar computational complexities.

- $\mathcal{EXP}$ is the class of problems solvable in exponential time.
- $\mathcal{PSPACE}$ is the class of problems solvable in polynomial space.
- $\mathcal{BQP}$ is the class of problems solvable by a quantum computer in polynomial time with reasonably low chance of error.

Definition

$\boldsymbol{\mathcal{P}}$ is the class of decision problems solvable in polynomial time.

Complexity Classes

Definition

$\mathcal{NP}$ is the class of decision problems for which a "yes" answer is checkable in polynomial time.

- A decision problem is not necessarily easy to *solve*.
- A solution to a decision problem is often easy to *check*.

Example

Recall the Traveling Salesperson Problem:

- Checking a solution to optimization TSP requires comparing it to all other Hamiltonian cycles.
- Checking a solution to decision TSP requires verifying that it is Hamiltonian and has weight $\leq k$.

Complexity Classes

Definition

$\boldsymbol{\mathcal{NP}}$**-Hard** is the class of problems at least as hard as every other problem in $\mathcal{NP}$.

Definition

$\boldsymbol{\mathcal{NP}}$**-Complete** is the class of problems in both $\mathcal{NP}$ and $\mathcal{NP}$-Hard.

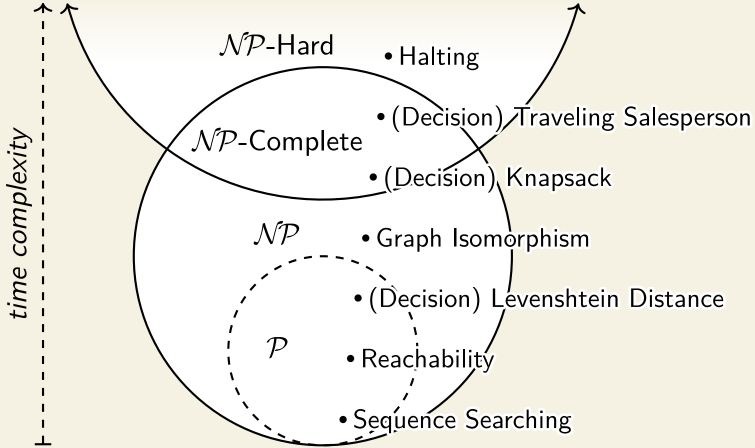
- Problems in $\mathcal{NP}$ are easy to check; problems in $\mathcal{NP}$-Hard are hard to solve.
- Problems in $\mathcal{NP}$-Complete have solutions which can be *checked* efficiently but not *found* efficiently.

Corollary

If a problem is easy to solve, then it is easy to check: $\mathcal{P} \subseteq \mathcal{NP}$

$\mathcal{P}$ and $\mathcal{NP}$

Example



$\mathcal{P}$ and $\mathcal{NP}$

Example *(cont.)*

The decision version of TSP is $\mathcal{NP}$-Complete:

- Finding a Hamiltonian cycle of weight $\leq k$, or determining that one does not exist, has complexity $O\big(|V|^2 \cdot 2^{|V|}\big)$.
- Checking that a cycle is Hamiltonian and of weight $\leq k$ has complexity $O(|V|)$.

- If $\mathcal{P} = \mathcal{NP}$, then there exist polynomial time algorithms to solve $\mathcal{NP}$-Complete problems — we haven't found them yet.
- If $\mathcal{P} \neq \mathcal{NP}$, then there exist $\mathcal{NP}$-Complete problems which we will never be able to solve in polynomial time.

Reductions

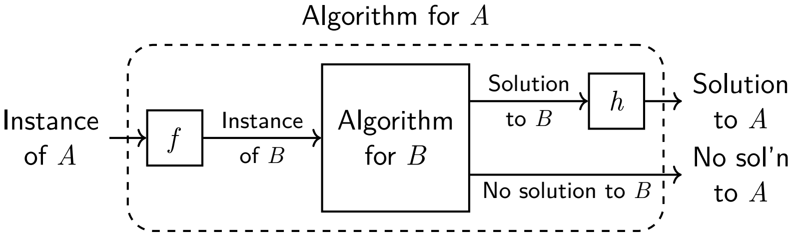
Reductions

Definition

A **reduction** from a problem A to a problem B consists of:

- An algorithm f, transforms instances of A into those of B
- An algorithm h, transforms solutions of B into those of A

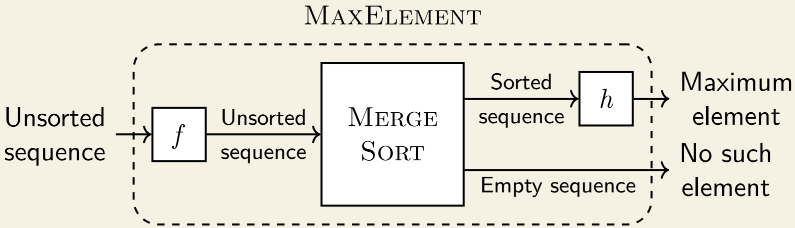
...such that no solution to B implies no solution to A.



Reductions

Example

Consider reducing Maximum Element to Sorting:



Where:

- f is the identity function.
- MergeSort sorts the sequence in descending order.
- h returns the first element of the sorted sequence.

Reductions

- A reduction uses an algorithm for B to solve A.
- It is typically required that the complexities of f and h be negligible compared to that of the algorithm for B.

Corollary

If A is reducible to B, then B is at least as hard as A.

- If B were easier than A, then the algorithm for B could be used to solve A, making A exactly as hard as B.

Example *(cont.)*

Suppose there existed a $\textsc{MagicSort}$ with complexity $< O(n)$. Then $\textsc{MaxElement}$ would also have complexity $< O(n)$.

The Boolean Satisfiability Problem

Definition

A **proposition** is either true or false, but not both.

- Propositions may be combined using **logical operators**:
  - Negation ("$\neg p$")
  - Conjunction ("$p \wedge q$")
  - Disjunction ("$p \vee q$")
  - ...
- Propositions are closed under these operations.

Example

Given that $p \equiv F$, $q \equiv F$, and $r \equiv T$, $(p \vee \neg q) \wedge r \equiv T$.

Definition

A proposition is **satisfiable** if and only if there exists an assignment of truth values to its variables such that it is true.

The Boolean Satisfiability Problem

Definition

A proposition in **conjunctive normal form**, or “CNF”, is a conjunction of clauses, where each clause is a disjunction of literals.

- A sub-proposition, typically parenthesized, is called a “clause”.
- A variable identifier, optionally negated, is called a “literal”.
- In other words, a proposition in CNF is an “‘and’ of ‘or’s”.

Example

$(p \vee \neg q) \wedge (\neg p \vee q)$ is in CNF.

Example

$(p \wedge \neg q) \vee (\neg p \wedge q)$ is not in CNF.

The Boolean Satisfiability Problem

Consider the following problem:

- Given a proposition in CNF with exactly 3 literals per clause, determine whether or not it is satisfiable.

Example

Given:

$$(p \vee q \vee r) \wedge (\neg p \vee r \vee \neg p) \wedge (\neg r \vee \neg r \vee \neg r)$$

...$p \equiv F$, $q \equiv T$, $r \equiv F$ is a satisfying assignment.

The Cook-Levin Theorem

The Boolean Satisfiability Problem, or "SAT", is $\mathcal{NP}$-Complete.

Corollary

The 3-Satisfiability Problem, or "3-SAT", is $\mathcal{NP}$-Complete.

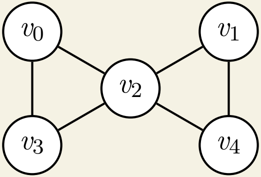
The Clique Problem

Consider the following problem:

- Given a graph, find a maximum **clique**, a subgraph containing exactly one edge between every distinct pair of vertices.

Example

Given:



...$\{v_0, v_2, v_3\}$ induces a maximum clique.

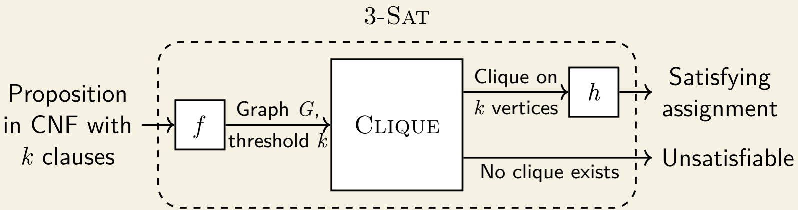
The Clique Problem

Theorem

The (Decision) Clique Problem is $\mathcal{NP}$-Complete.

Example

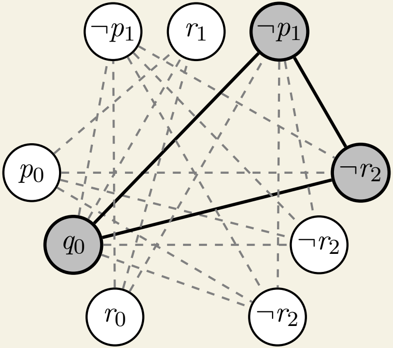
Consider reducing 3-SAT to Clique:



The Clique Problem

Example *(cont.)*

Given $(p \vee q \vee r) \wedge (\neg p \vee r \vee \neg p) \wedge (\neg r \vee \neg r \vee \neg r)$:



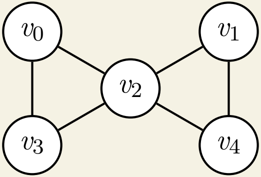
The Vertex Cover Problem

Consider the following problem:

- Given a graph, find a minimum **vertex cover**, a subset of vertices incident to every edge in a graph.

Example

Given:



...$\{v_0, v_1, v_2\}$ is a minimum vertex cover.

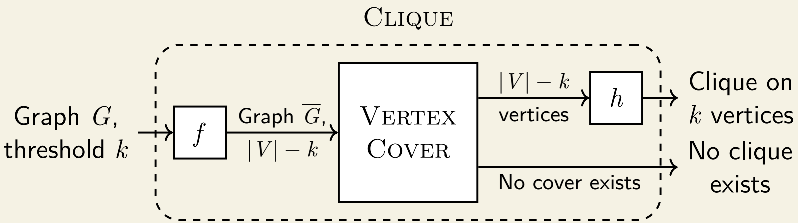
The Vertex Cover Problem

Theorem

The (Decision) Vertex Cover Problem is $\mathcal{NP}$-Complete.

Example

Consider reducing Clique to Vertex Cover:



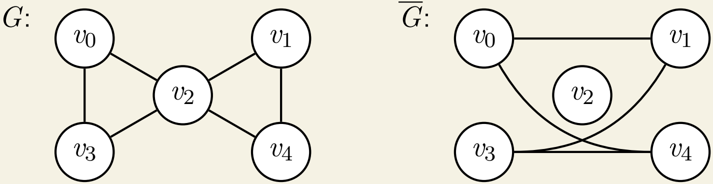
The Vertex Cover Problem

Definition

The **complement** of a simple graph $G = (V, E)$, denoted $\overline{G}$, is a simple graph where $(u, v) \in \overline{E}$ if and only if $(u, v) \notin E$.

- Vertices that are not in a vertex cover cannot be connected.
- $S \subseteq V$ is a vertex cover of $\overline{G}$ iff $V - S$ induces a clique of G.

Example *(cont.)*



Karp's Twenty-One $\mathcal{NP}$-Complete Problems

Example

VERTEXCOVER can be used to solve the Clique Problem, and CLIQUE can be used to solve the Vertex Cover Problem.

- All of the following problems are reducible to one another:
  - Boolean Satisfiability
  - 3-Satisfiability
  - Clique
  - Vertex Cover
  - k-Colorability
  - Hamiltonian Cycle
  - Knapsack
  - ...
- If any of these problems can be solved in polynomial time, then they *all* can (and, thus, it will be the case that $\mathcal{P} = \mathcal{NP}$).

Approximations

Approximation Algorithms

- Assuming that $\mathcal{P} \neq \mathcal{NP}$, *none* of the $\mathcal{NP}$-Complete problems can be solved in polynomial time.

Definition

An **approximation algorithm** is one that efficiently approximates a solution to a problem.

- Approximations are applied to $\mathcal{NP}$-Hard optimization problems.
  - Problems not in $\mathcal{NP}$-Hard are already relatively easy to solve.
  - Decision problems have inapproximable solutions.
- Approximations are typically compared to optimal algorithms.

Approximation Ratios

Definition

An algorithm has an **approximation ratio** of ρ if and only if an approximate solution A is at most ρ times worse than the optimal solution $\textsc{Opt}$.

- A **ρ-approximation** is an algorithm with a ratio of ρ.
  - For minimization problems, $\textsc{Opt} \leq A \leq \rho \cdot \textsc{Opt}$.
  - For maximization problems, $^1/_\rho \cdot \textsc{Opt} \leq A \leq \textsc{Opt}$.
- A 1-approximation algorithm is an optimal algorithm.

Example

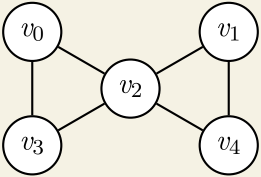
Simply returning every vertex in a graph $G = (V, E)$ yields a $|V|$-approximation of Vertex Cover.

Approximating Vertex Cover

- A **matching**, a subset of edges such that no two edges share an endpoint, can be found greedily in polynomial time.
- The endpoints of a maximal matching form a vertex cover.

Example

Given:



...$\{(v_0, v_2), (v_1, v_4)\}$ is a maximal matching, and $\{v_0, v_1, v_2, v_4\}$ is a vertex cover.

Approximating Vertex Cover

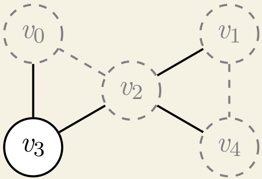
$\textsc{ApproximateVertexCover}(G = (V, E))$

Input: A graph G
Output: A vertex cover of G

```
let S be ∅
for all e = (u, v) ∈ E do
    if u ∉ S and v ∉ S then
        let S be S ∪ {u, v}
return S
```

Approximating Vertex Cover

Example



1. Add (v_0, v_2).
2. Skip (v_0, v_3).
3. Skip (v_1, v_2).
4. Add (v_1, v_4).
5. Skip (v_2, v_3).
6. Skip (v_2, v_4).
7. Return $\{v_0, v_1, v_2, v_4\}$.

Note than an optimal vertex cover has cardinality 3.

Approximating Vertex Cover

Example

- The optimal solution includes at least 1 vertex for each edge.
- The approximation includes exactly 2 vertices for each edge.
- $\textsc{Opt} \leq A \leq 2 \cdot \textsc{Opt}$

$\textsc{ApproximateVertexCover}$ is a 2-approximation.

- Each edge is considered exactly once.
- Each vertex is considered at most $\deg(v)$ times.
- Vertices are collectively considered $2\,|E|$ times.

$\textsc{ApproximateVertexCover}$ has complexity $O(|E|)$.

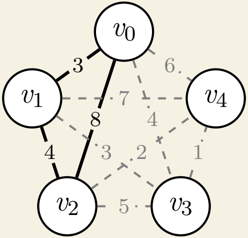
Approximating Metric TSP

Consider the following problem:

- Given a complete graph with weights satisfying the **triangle inequality**, find the Hamiltonian cycle of minimum weight.

Example

Given:



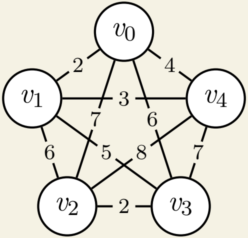
...the edge (v_0, v_2) violates the triangle inequality: $8 \not\leq 3 + 4$.

Approximating Metric TSP

- A Hamiltonian cycle less one edge forms a spanning tree.
- An MST can be found greedily in polynomial time.

Example

Given:



...a Hamiltonian cycle of minimum weight is $(v_0, v_1, v_3, v_2, v_4, v_0)$, and $(v_0, v_1, v_3, v_2, v_4)$ defines a spanning tree.

Approximating Metric TSP

$\textsc{ApproximateMetricTSP}(G = (V, E))$

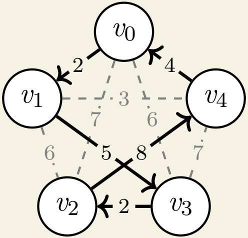
Input: An complete, weighted graph G with weights satisfying the triangle inequality

Output: A Hamiltonian cycle in G

let T be $\textsc{MinimumSpanningTree}(G)$
for all $v \in V$ **do**
    let v be "unexplored"
let s be any vertex in V and P be the vertices of $\textsc{Explore}(T, s)$, sorted in ascending order by previsit number
return $P + (s)$

Approximating Metric TSP

Example



1. Construct an MST: $\{(v_0, v_1), (v_2, v_3), (v_1, v_4), (v_1, v_3)\}$
2. Traverse tree edges: $(v_0, v_1, v_3, v_2, v_3, v_1, v_4, v_1, v_0)$
3. Bypass duplicate vertices: $(v_0, v_1, v_3, v_2, v_4, v_0)$

Note that an optimal Hamitonian cycle has weight 21.

Approximating Metric TSP

Example

- A Hamiltonian cycle cannot be lighter than an MST.
- By the triangle inequality, bypassing cannot increase weight.
- $\textsc{Mst} \leq \textsc{Opt} - w_e \leq \textsc{Opt} \leq A \leq 2 \cdot \textsc{Mst}$

$\textsc{ApproximateMetricTSP}$ is a 2-approximation.

- Constructing an MST has complexity $O(|E| \log |E|)$.
- Exploring a spanning tree has complexity $O(|V|)$.
- Preordering vertices while exploring adds negligible complexity.

$\textsc{ApproximateMetricTSP}$ has complexity $O(|E| \log |E|)$.

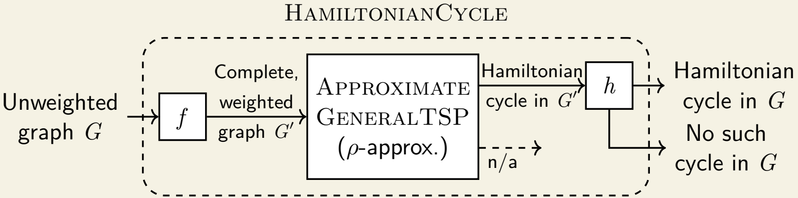
Approximating General TSP

Theorem

Approximating general TSP with any constant ratio ρ is $\mathcal{NP}$-Hard.

Example

Consider reducing Hamiltonian Cycle to Approximate TSP:



Approximating General TSP

Example *(cont.)*

Given a Hamiltonian graph G and $\rho = 2$: