Approximation Algorithms

Martin Böhm (University of Bremen)

Lecture 2, April 4, 2019

Set Cover and Hardness

Greedy and Approximation Algorithms

Definition An α-approximation algorithm for an optimization problem is an algorithm that

- runs in polynomial time and
- computes for any instance of the problem a solution,
- whose value is within a factor of α of the optimal solution.

Greedy Algorithm

- iteratively constructs a solution, at each iteration making locally best choice
- typically simple, efficient (polynomial) run time

Example III: Set Cover

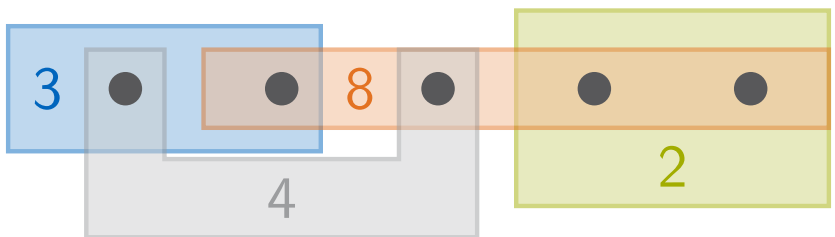
The Set Cover problem

Input: A universe $U = \{1, 2, \ldots, n\}$.
A list of sets $S_1, \ldots, S_m$, all $S_i \subseteq \{1, \ldots, n\}$.
Each set S_i has associated weight $w_i \geq 0$.

Output: A cover $C \equiv$ a selection of sets ($C \subseteq \{1, \ldots, m\}$)
which *covers* the entire universe: $\bigcup_{j \in C} S_j = U$.

Goal: Find a cover C minimizing the total weight $\sum_{j \in C} w_j$.

Equivalent: Minimize $\sum_{i=1}^{m} w_i x_i$, where $x_i = 1$ if i selected,
$x_i = 0$ otherwise.



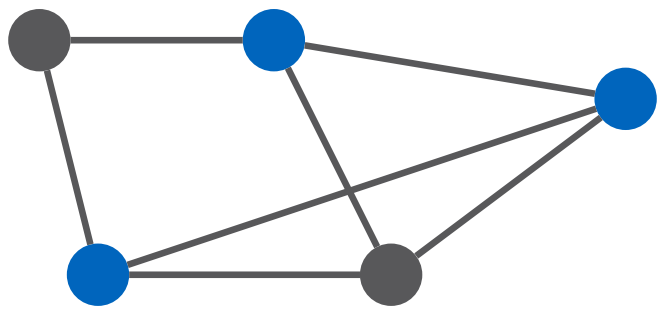
Set Cover: "A universe is covered by sets."

Special case: Weighted Vertex Cover

Input: An undirected, weighted graph $G = (V, E)$.
Every vertex $v \in V$ has an associated weight $w_v \geq 0$.

Output: A cover $C \subseteq V$ such that every edge $uv \in E$ has either u or v *(or both)* in C.

Goal: Find a cover C minimizing the total weight $\sum_{v \in C} w_v$.



Vertex Cover: "Edges are being covered by vertices."

Greedy algorithm

1. While $\exists$ uncovered element:
2. Consider a set $\hat{S}_j$ to only hold uncovered elements:

$$\widehat{S}_j \equiv S_j \setminus \bigcup_{k \in C} S_k.$$

3. Choose j minimizing $\frac{w_j}{|\widehat{S}_j|}$ and add it to C.

Theorem

The Greedy Algorithm is an H_n-approximation for Set Cover.

$$n := |U|,\ H_n := \sum_{i=1}^{n} \tfrac{1}{i} \approx \log n$$

n_k: number of uncovered elements at the beginning of iteration k
S_j: set selected in iteration k

Lemma

For every iteration k: $\quad w_j \leq \dfrac{n_k - n_{k+1}}{n_k} \operatorname{OPT}.$

Definition

An analysis that an algorithm is an α-approximation algorithm is tight when there is an instance I for which the optimum value OPT_I satisfies

$$\alpha \cdot OPT_I = ALG_I.$$

Example: The analysis for the greedy algorithm for List Scheduling which shows $(2 - 1/m)$-approximation is tight.

👩: Why have you not solved the problem exactly?

NP-completeness

NP-hardness

Definition (Decision problem)

A problem is a decision problem when the answer is either `true` or `false` for all instances.

Examples:

1. Given a set of inequalities, is there a solution with all integer values?
2. Given a Boolean formula (say $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$), is there an assignment of 0/1 values such that the formula evaluates to 1?
3. Given a set of jobs J with processing times, can you schedule them on m machines with makespan at most M?

Fact: Only decision problems can be NP-hard or NP-complete.

Definition (Decision problem)

A problem is a decision problem if the answer is either `yes` or `no` for all instances.

Fact: Only decision problems can be NP-hard or NP-complete.

Definition (Optimization problem)

A problem is an optimization problem if the input has both constraints and an objective function. Any solution which satisfies the constraints can be called feasible, but the goal is usually to find the optimal feasible solution.

Also fact: Approximation algorithms solve only optimization problems.

Q: Can we find a related optimization problem for a decision one?
A: Oftentimes yes, but not always.

Examples:

- Find the largest vertex cover in a graph. $\approx$ Decide if the largest vertex cover is $\leq k$.
- Color the graph with the least amount of colors. $\approx$ Decide if the graph can be colored with at most k colors.

But also:

- Given a system of equations, decide if it has a solution. $\approx$??
- Given a set of equations, satisfy as many as possible. $\approx$ Given a set of equations, decide if at least k can be satisfied at the same time.
- Given a graph G which is guaranteed to be 3-colorable, color it with the least amount of colors. $\approx$??

P and NP

Definition

The class P is a class of all *decision* problems which have a polynomial-time algorithm correctly deciding them.

Definition

The class NP is a class of all *decision* problems which can be verified in polynomial time.

Verification $\equiv$ existence of an algorithm V such that:

1. V runs in polynomial time.
2. For every `yes` instance I, there exists a polynomial-size (polynomial with respect to I) advice A such that

$$V(I, A) = \texttt{yes}.$$

3. For every `no` instance I, and for any polynomial size binary string A, $V(I, A)$ always reports `no`.

Reductions between problems

Sometimes we can solve a (decision) problem by converting (reducing) it to another problem.

Example: Solving Weighted Vertex Cover with an algorithm that solves Set Cover: just rewrite the instance of VC to be an instance of SC.

Definition

A decision problem D has a polynomial-time reduction to a decision problem E if there exist a poly-time algorithm A such that:

1. Given an instance I_D of the problem D, $A(I_D)$ outputs I_E, an instance of the problem E.
2. The instance I_D has a yes answer if and only if the instance $I_E = A(I_D)$ has a yes answer.

NP-Completeness

Definition

A decision problem D is NP-complete if:

1. D belongs to NP *(D can be verified quickly)*.
2. For any problem $C \in NP$, C can be reduced to D. *(D is the hardest problem in the whole class NP.)*

Theorem (Cook-Levin theorem)

There exists an NP-complete problem.

Note: There are actually many.

Many NP-complete problems

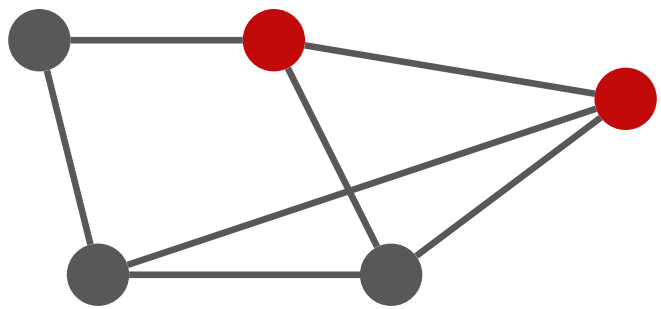
- Satisfiability: Given a Boolean formula F, decide if it has an 0/1 assignment that evaluates the formula to `true`.
- Clique: Given a graph G and number k, decide if there is a subset of at least k vertices such that every vertex is connected to every other vertex in this set.
- Independent Set: Given a graph G and number k, decide if there is a subset of at least k vertices such that none of the vertices is connected to anyone else in this set.
- Set Cover, Vertex Cover, Scheduling, k-center ...

Example: Dominating Set

Decision problem Dominating Set:

Input: An undirected, unweighted graph $G = (V, E)$, an integer k.

Decide: Is there a set of at most k vertices $D \subseteq V$, such that for every vertex $v \in V$ it holds that (a) $v \in D$ or (b) a neighbor of v is in D?



Theorem

DOMINATING SET *is NP-complete.*

Decision problem DOMINATING SET:

Input: An undirected, unweighted graph $G = (V, E)$, an integer k.

Decide: Is there a set of at most k vertices $D \subseteq V$, such that for every vertex $v \in V$ it holds that (a) $v \in D$ or (b) a neighbor of v is in D?

Decision problem VERTEX COVER:

Input: An undirected, unweighted graph $G = (V, E)$, an integer l.

Decide: Is there a cover $C \subseteq V$ of size at most l such that every edge $uv \in E$ has either u or v *(or both)* in C?

Proof steps:

1. Prove that DOMINATING SET lies in NP.
2. Reduce from VERTEX COVER to DOMINATING SET.

👩: Yes, the problem is NP-complete, but why is the approximation ratio α so big?

Hardness of approximation

Hardness of approximation

Definition

An optimization problem is NP-hard to approximate within factor α if an α-approximation algorithm can be used to solve some NP-complete problem.

Basic approach: If we wanted to prove that K-CENTER is hard to approximate within factor less than 2:

- Take some NP-complete decision problem P.
- Find a reduction from P to K-CENTER such that `yes`-instances of P are mapped to instances of K-CENTER with optimal radius 1 . . .
- . . . and `no`-instances are only mapped to instances with optimal radius ≥ 2.
- Then, any 1.99-approximation algorithm is able to distinguish `yes`-instances from `no`-instances.

Hardness of K-CENTER

Theorem

K-CENTER *is NP-hard to approximate within factor less than* 2.

Reduce from DOMINATING SET:

Input: An undirected, unweighted graph $G = (V, E)$, an integer k.

Decide: Is there a set of k vertices $D \subseteq V$, such that for every vertex $v \in V$ it holds that (a) $v \in D$ or (b) a neighbor of v is in D?