

APPROXIMATION ALGORITHMS

- NPC decision problems often correspond to hard optimization problems of huge practical impact: we cannot give up on them!

Different approaches:

1. Non-polynomial algorithms:

sometimes suitable for "small" instances. Interesting sub-category:

pseudopolynomial algorithms:

complexity polynomial in the values (and NOT the size of their encoding)

EXAMPLE: Using dynamic programming we can obtain an algorithm for SS with running time $O(|S| \cdot t)$

(homework: |S| t - boolean table A:

$$A[i,j] = 1 \Leftrightarrow \exists S' \subseteq \{s_1, \dots, s_t\} : \sum_{s \in S'} s = j$$

$i \in |S|, 0 \leq j \leq t$. Solution

in $A[|S|, t]$. Work out recurrence)

$|S| \cdot t$ is polynomial in the value t but NOT in the length of its encoding $O(\log t)$

2. Restriction of the instance set : solve the problem on a subset of instances for which there is a polynomial algorithm

EXAMPLE : 2-CNF-SAT $\in P$

TREE-VERTEX-COVER $\in P$

SMALL-TARGET SUBSET SUM $\in P$

$$(t \in O(\log^k(\sum_{i \in S})))$$

3. "intelligent" exhaustive search (O.R. approach) : e.g. branch-and-bound : try to get optimal solution while avoiding to examine all feasible solutions (pruning strategies).
Heuristic methods : no upper bound on worst-case running time -

Other heuristics : local-search, tabu-search, simulated annealing : may return nonoptimal solution (local optimum) (\rightarrow see OR2)

4. APPROXIMATION ALGORITHMS : return
a feasible solution whose cost is
within a given error margin from
the optimal solution in worst-case
polynomial time

EXAMPLE : Return, in linear time, a
vertex cover of size at most twice
the size of the minimum vertex
cover

3 vs 4 : We want hard guarantees on
time and solution quality
(Sometimes a solution obtained through
4. is well as a starting solution
for the heuristics of 3.)

Let Π be an optimization problem with
positive integer cost function :

$$\Pi \subseteq \mathcal{S} \times \mathcal{I}, \quad \forall i \in \mathcal{I} : \mathcal{I}(i) = \{ s \in \mathcal{S} : i \in \Pi(s) \}$$

$c : \mathcal{S} \rightarrow \mathbb{N} - \{0\}$, and let

$$s^*(i) = \arg \min_{s \in \mathcal{S}(i)} c(s)$$

DEF: Let A_π be an algorithm such that $\forall i \in J : A_\pi(i) \in S^*(i)$. We say that A_π is a $g(u)$ -approximation algorithm for Π if $\forall i \in J$, $i|l=u$:

$$\max \left\{ \frac{c(S^*(i))}{c(A_\pi(i))}, \frac{c(A_\pi(i))}{c(S^*(i))} \right\} \leq g(u)$$

approximation ratio

OBSERVATION 1 : The above max is always ≥ 1 :
 $\Rightarrow g(u) \geq 1, \forall u$. Better approximation achieved for $g(u)$ closer to 1. ($\forall u : g(u)=1$: EXACT ALGORITHM)

OBSERVATION 2 : For maximization problems the above max is always the first term, and for minimization problems the second. Equivalently:

$$c(A_\pi(i)) \begin{cases} \geq \frac{c(S^*(i))}{g(u)} & \text{MAX. PROBLEMS} \\ & \text{(fraction } g(u) \text{)} \\ \leq c(S^*(i)) \cdot g(u) & \text{MIN. PROBLEMS} \\ & \text{(factor } g(u) \text{)} \end{cases}$$

An approximation algorithm finds feasible solutions of "controlled quality". Also, it provides bounds on the optimal solution

In particular :

$$c(s^*(i)) \leq c(A_{\pi}(i)) g(n) \text{ MAX.}$$

upper bound on $c(s^*(i))$

$$c(s^*(i)) \geq \frac{c(A_{\pi}(i))}{g(n)} \text{ MIN}$$

lower bound on $c(s^*(i))$

DEF We say that Π is $g(n)$ -approximable in polynomial time if there exists a $g(n)$ -approximation algorithm A_{Π} for Π running in polynomial time.

Hard optimization problems are not all equal w.r.t. approximation: some are $g = O(1)$ -approximable (in poly. time), some only feature $g(n)$ with $\lim_{n \rightarrow \infty} g(n) = \infty$ -

We can often prove that if $P \neq NP$ there cannot exist a poly-time $g(n)$ -approximation algorithm for some function $g(n)$. (INAPPROXIMABILITY RESULTS).

EXAMPLE : If $P \neq NP$ there cannot exist a $g(|V|) = \Theta(|V|^{1-\epsilon})$ poly-time approximation algorithm for MAX-CLIQUE, for any fixed ϵ
 [Arora et al. 1990]

there are also "easier hard" problems which admit very "precise" approximation algorithms :

DEF : An approximation scheme for an optimization problem Π is a two-input algorithm

$$A_{\Pi}(i, \epsilon) : i \in \mathbb{J}, \epsilon > 0$$

such that, for any fixed ϵ , $A_{\Pi}(i, \epsilon)$ is a $g(u) = (1+\epsilon)$ -approximation algorithm for Π .

DEF. An Approximation Scheme is Polynomial-Time (PTAS) if, for any fixed ϵ , $A_{\Pi}(i, \epsilon)$ runs in time polynomial in $N = |i|$

Observe that under the above definition, running times such as $O(n^{1/\epsilon})$

are polynomial in n (for fixed ϵ)

This is not completely satisfactory:
e.g., if I want to reduce the "error" ϵ by a factor k :

$$O(n^{1/\epsilon}) \rightsquigarrow O(n^{1/(\epsilon/k)}) = O((n^{1/\epsilon})^k).$$

Time exponential in the improvement

It would be desirable that a constant-factor improvement in quality yield a constant-factor decrease in running time;

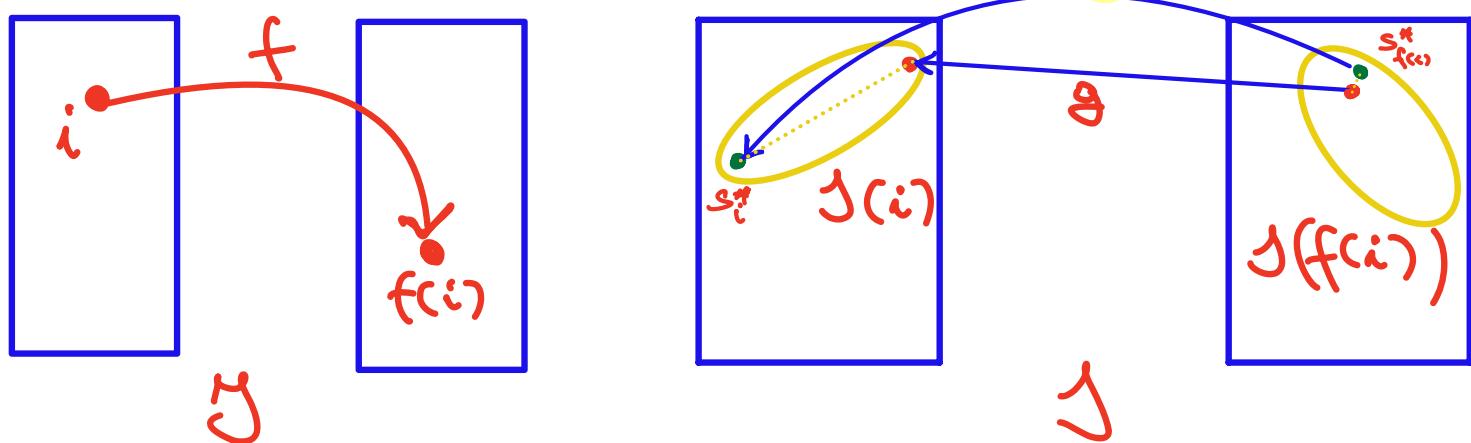
DEF An Approximation Scheme is Fully-Polynomial Time (FPTAS) if the running time of $A_{\pi}(i, \epsilon)$ is polynomial both in $n=1/l$ and $\frac{1}{\epsilon}$

$$\text{e.g. } T_{A_{\pi}}(n, \epsilon) = O\left(\frac{1}{\epsilon^2} n^3\right)$$

More

While optimization problems associated to NPC problems are all equivalent w.r.t. exact solution, they behave differently w.r.t. approximation.

Intuition : The idea of reducibility can be extended to optimization problems using an extra function g , mapping each element of $\mathcal{I}(f(i))$ into an element of $\mathcal{I}(i)$:



Function g maps optimal solutions in $\mathcal{I}(f(i))$ into optimal solutions in $\mathcal{I}(i)$, however the mapping does not necessarily preserve the approximation quality !

A TWO-APPROXIMATION ALGORITHM FOR (MINIMUM) VERTEX COVER

Optimization version: given $G = (V, E)$, undirected, determine its vertex cover $V^* \subseteq V$ ($\forall \{u, v\} \in E: (u \in V^*) \vee (v \in V^*)$) of minimum size.

We will prove that a simple greedy approach yields a good quality solution

IDEA : 1. GREEDY CHOICE : consider an arbitrary edge $\{u, v\}$ and ADD BOTH u and v to the vertex cover. (OBS: NOT INTUITIVE!)
2. CLEAR-VR : remove all covered edges from E : $E' = E - \{e \in E : \{u, v\} \cap e \neq \emptyset\}$

APPROX-VERTEX-COVER ($(G = (V, E))$)

$V' \leftarrow \emptyset$

$E' \leftarrow E$

while ($E' \neq \emptyset$) do

* select $\{u, v\} \in E'$, arbitrary * GC

$V' \leftarrow V' \cup \{u\} \cup \{v\}$

$E' \leftarrow E' - \{e \in E' : \{u, v\} \cap e \neq \emptyset\}$ c.v.

return V'

The algorithm correctly returns a vertex cover (I eliminate an edge only when it is covered by some selected node)

Running time: $O(|V| + |E|) = O(|E| \leq G = (V, E)|)$
(every time I select edge $\{u, v\}$, I remove the two adjacency lists of u and v .)

TOUGH PART OF THE ANALYSIS: approximation ratio.

Let A be the set of edges selected by $A_VC(G)$ at each iteration of the while loop.

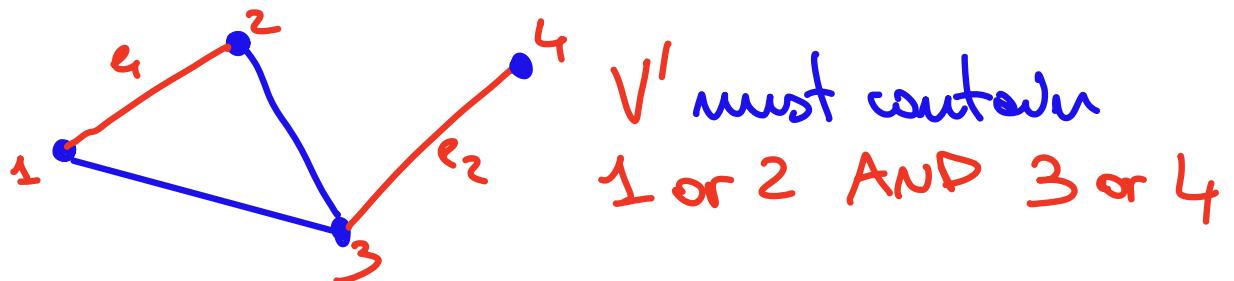
PROPERTY: $\forall e_1 \neq e_2 \in A: e_1 \cap e_2 = \emptyset$

The edges in A do not share endpoints (since at each iteration all edges sharing one endpoint with the selected edge are removed)

A set of edges A not sharing endpoints is called a matching

FACT: If a graph $G = (V, E)$ contains a matching $A \subseteq E$, then for any vertex-cover $V' \subseteq V$ we have $|V'| \geq |A|$

PROOF: There must be at least one endpoint of each edge e in A in V' , or otherwise e would not be covered.



COROLLARY: $|V^*| \geq |A|$

Algorithm A-V-C (G) returns a vertex cover V' , with $|V'| = 2|A|$

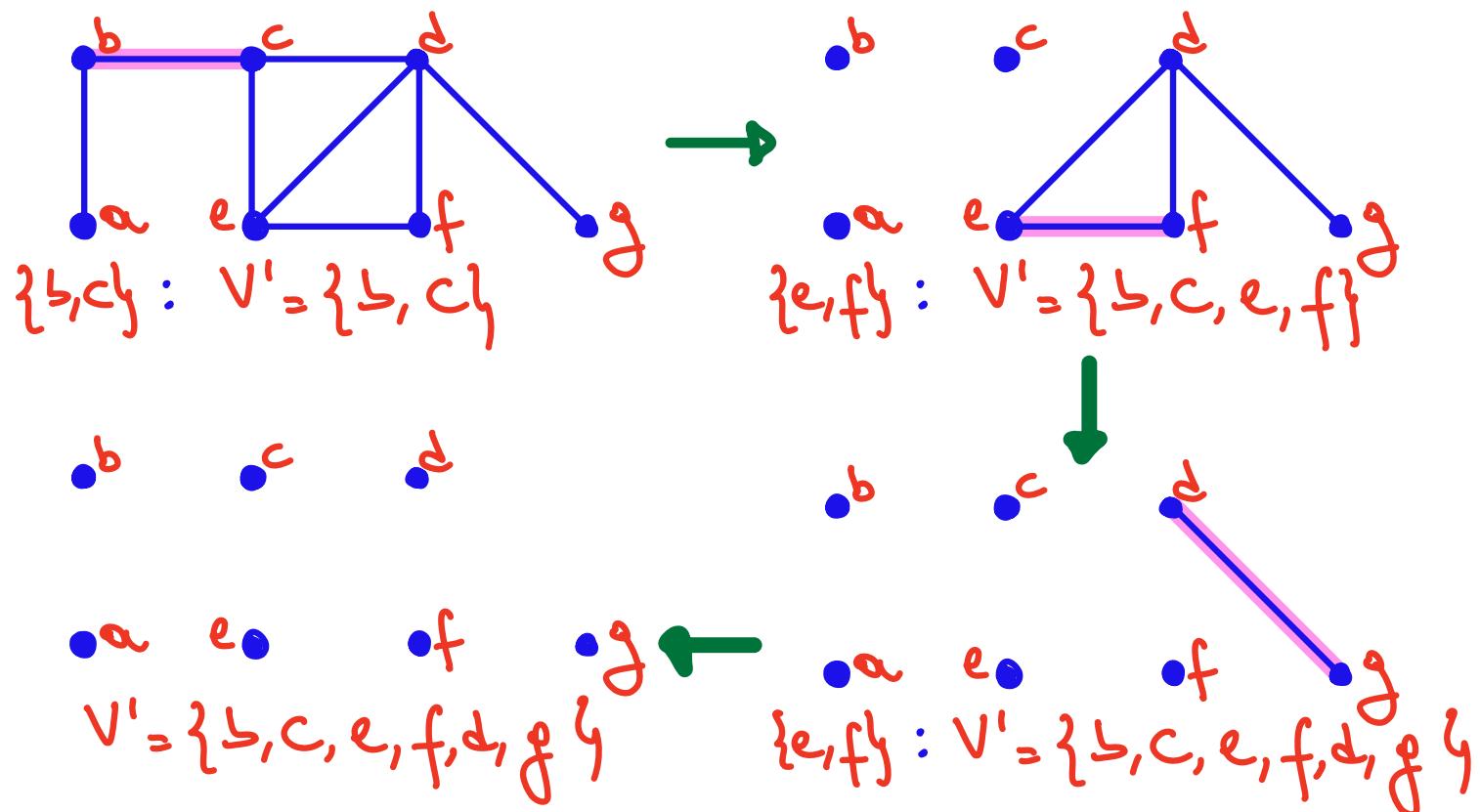
$$\text{We have } g = \frac{|V'|}{|V^*|} \leq \frac{2|A|}{|A|} = 2$$

IMPORTANT: To prove an upper bound to g , we always need:

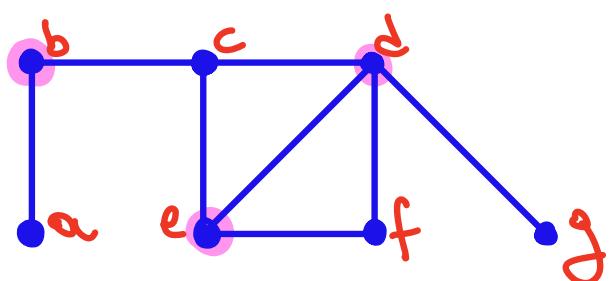
- a lower bound to $c(s^*)$ MINIMIZATION
- an upper bound to $c(s^*)$ MAXIMIZATION

We can show that the bound on the approximation ratio is tight by exhibiting an instance for which

$$c(A_{\pi}(i)) = g \cdot c(S^*(i)) \text{ (MN)}$$



$|V'| = 6$ but :



$$V^* = \{b, d, e\}$$

$\Rightarrow |V'| = 2|V^*|$ (tight approximation!)

LESSONS LEARNED: To get a bound on the approximation we need to establish explicit bounds (lower or upper) on $|CS^*|$.

For VC, we establish such a (lower) bound by relating $|V^*|$ (cannot be computed) to the size of an easily obtainable construct (a matching A of G). We will see that this approach can be used for other problems.

OBSERVATION: The matching A constructs implicitly by $A = V - C(G)$ is maximal, in the sense that $\forall e \in E : A \cup \{e\}$ is not a matching - (PROVE IT!)

EXERCISE: Prove that the set of all nodes forming the edges of a maximal matching are a vertex cover