

Dealing with NP-completeness

- Imagine that you must find an efficient algorithm for a particular problem, say as part of your job. After some time, you are still unable to discover such an algorithm. After further thought, you realize that the problem is NP-hard.
- It is extremely unlikely that you will find an efficient algorithm for this problem, since doing so would prove $\mathbf{P} = \mathbf{NP}$, a proposition that is not believed to be true.
- On the other hand, you cannot simply give up: you must find a solution of some kind.
- One solution is to change the problem you are trying to solve. There are many ways of doing this. In the case of optimization problems, one technique that is often successful is, instead of insisting on finding an optimal solution, to search for a solution that is “close” to optimal.

Example: Vertex-Cover revisited

- Consider the following problem VERTEX-COVER-OPT:

Instance: A graph $G = (V, E)$

Solution: A vertex cover C for G

Objective: Minimize $|C|$

- Clearly we can't find a polynomial time algorithm for VERTEX-COVER-OPT unless $\mathbf{P} = \mathbf{NP}$, since such an algorithm could be used to decide VERTEX-COVER in polynomial time.
- Instead of trying to find an optimal solution, we'll find a vertex cover whose size differs from the optimum by at most a constant factor.
- **Definition:** A matching in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that for all $e_1 \neq e_2 \in M$, $e_1 \cap e_2 = \emptyset$; i.e. a set of edges that do not share any endpoints. A matching M is maximal if for all $e \in E \setminus M$, $M \cup \{e\}$ is not a matching.
- It is easy to find a maximal matching in polynomial time: just keep taking edges that maintain the matching property, until there are no more such edges.
- Now consider the following algorithm for VERTEX-COVER-OPT:
 1. Find a maximal matching M
 2. Return $C = \{\text{endpoints of edges in } M\}$
- **Claim:** C is a vertex cover.

Input: $G = (V, E)$

- 1: $M \leftarrow \emptyset$
- 2: **while** $E \neq \emptyset$ **do**
- 3: choose $e \in E$ arbitrarily
- 4: $M \leftarrow M \cup \{e\}$
- 5: $E \leftarrow E \setminus \{e' \in E \mid e' \cap e \neq \emptyset\}$
- 6: **end while**
- 7: return M

Algorithm 1: Algorithm to find a maximal matching

- Proof: Let $e \in E$. If $e \in M$ then e is covered, since C include all endpoints of edges in M . If $e \notin M$, then there is an edge $e' \in M$ such that $e' \cap e \neq \emptyset$, since M is maximal. Say $e' \cap e = \{u\}$. Then $u \in C$, so e is covered.
- Is C an optimal vertex cover? No: consider, for example, the complete bipartite graph $K_{n/2, n/2}$ (i.e. the graph on n vertices consisting of two sets U_1, U_2 of size $n/2$ each, which contains all edges between U_1 and U_2 and no other edges). Then the optimal solution OPT takes every vertex in U_1 (or every vertex in U_2), so $|\text{OPT}| = n/2$. On the other hand, the algorithm we described finds a perfect matching M with $|M| = n/2$, and takes both endpoints of every edge in M , so it takes all n vertices in $U_1 \cup U_2$, and thus $|C| = 2 \cdot |\text{OPT}|$.
- In the above example, the vertex cover differed from OPT by a factor of 2. We can show that the algorithm never does worse than this: for every input graph, the solution C returned by the algorithm satisfies $|C| \leq 2|\text{OPT}|$.
- **Lemma:** Let $G = (V, E)$ be a graph, and let OPT be a minimum vertex cover for G . Then for every matching M of G , $|\text{OPT}| \geq |M|$.
- Proof: Since OPT is a vertex cover, it must cover every edge in M . Since M is a matching, no vertex can cover more than one edge in M , thus $|M|$ vertices are necessary to cover all of these edges.
- **Lemma:** Let $G = (V, E)$ be a graph, let OPT be a minimum vertex cover for G , and let C be the solution returned by the algorithm. Then $|C| \leq 2 \cdot |\text{OPT}|$.
- Proof: The solution C satisfies $|C| = 2|M|$, where M is the matching found by the algorithm. By the above lemma, $|M| \leq |\text{OPT}|$. So $|C| = 2|M| \leq 2 \cdot |\text{OPT}|$.
- This algorithm is called a 2-approximation for VERTEX-COVER-OPT. Since it runs in polynomial time, we have the following theorem.
- **Theorem:** There is a polynomial-time, 2-approximation algorithm for VERTEX-COVER-OPT.

Approximation Algorithms

- **Definition:** An optimization problem Π is specified by
 - A set I_Π of instances [in the case of VERTEX-COVER-OPT, the instances are graphs]
 - For each instance $I \in I_\Pi$, a set $S_\Pi(I)$ of feasible solutions [e.g. vertex covers]
 - An objective function w that maps $S_\Pi(I) \mapsto \mathbb{R}$ for each $I \in I_\Pi$ [e.g. $w(C) = |C|$]
 - An optimization direction: either minimize or maximize [e.g. minimize]

If Π is an optimization problem, then L_Π is defined as:

$$L_\Pi = \begin{cases} \{\langle I, k \rangle \mid I \in I_\Pi, w(\text{OPT}_\Pi(I)) \leq k\}, & \text{if } \Pi \text{ is a minimization problem} \\ \{\langle I, k \rangle \mid I \in I_\Pi, w(\text{OPT}_\Pi(I)) \geq k\}, & \text{if } \Pi \text{ is a maximization problem} \end{cases}$$

where $\text{OPT}_\Pi(I)$ denotes an optimal solution for instance I .

- **Fact:** If Π is an optimization problem and there is a polynomial-time algorithm for Π , then $L_\Pi \in \mathbf{P}$.
- **Definition:** An optimization problem Π is called **NP-hard** if L_Π is **NP-hard**.
- **Definition:** For $\alpha \geq 1$, an algorithm A is called an α -approximation algorithm for a minimization problem Π if for every instance $I \in I_\Pi$, A outputs a solution $C \in S_\Pi(I)$ satisfying $w(C) \leq \alpha \cdot w(\text{OPT}_\Pi(I))$.
- **Definition:** For $\beta \leq 1$, an algorithm A is called a β -approximation for a maximization problem Π if for every instance $I \in I_\Pi$, A outputs a solution $C \in S_\Pi(I)$ satisfying $w(C) \geq \beta \cdot w(\text{OPT}_\Pi(I))$.

Example: Set-Cover revisited

- Consider the following problem WEIGHTED-SET-COVER-OPT:

Instance: $S_1, \dots, S_m \subseteq U$, s.t. $\bigcup_{i=1}^m S_i = U$, and $w : \{S_1, \dots, S_m\} \mapsto \mathbb{R}$

Solution: A set $I \subseteq \{1, \dots, m\}$ such that $\bigcup_{i \in I} S_i = U$

Objective: Minimize $\sum_{i \in I} w(S_i)$

- Note that the corresponding decision problem WEIGHTED-SET-COVER is **NP-hard**: to reduce from SET-COVER, just assign every set a weight of 1.
- Consider algorithm 2 below.

Input: $S_1, \dots, S_m \subseteq U$, $\bigcup_{i=1}^m S_i = U$, and $w : \{S_1, \dots, S_m\} \mapsto \mathbb{R}$

- 1: $C \leftarrow \emptyset$
- 2: $I \leftarrow \emptyset$
- 3: **while** $C \neq U$ **do**
- 4: Choose $i \in \{1, \dots, m\}$ that minimizes $w(S_i)/|S_i \setminus C|$
- 5: For each $e \in S_i \setminus C$, set $\text{price}(e) = w(S_i)/|S_i \setminus C|$
- 6: $I \leftarrow I \cup \{i\}$
- 7: $C \leftarrow C \cup S_i$
- 8: **end while**
- 9: return I

Algorithm 2: Approximation algorithm for WEIGHTED-SET-COVER-OPT

- At each stage, the algorithm chooses the most cost-efficient set: i.e. the set that covers uncovered elements with the lowest cost per element.
- **Definition:** $H(n) = \sum_{k=1}^n 1/k$ is called the n -th harmonic number
- **Fact:** $H(n) \in \Theta(\log n)$
- **Theorem:** Algorithm 2 is an $H(n)$ -approximation algorithm for WEIGHTED-SET-COVER-OPT, where $n = |U|$.
- **Proof:** Number the elements of U e_1, \dots, e_n in the order they are covered by the algorithm, breaking ties arbitrarily. For $I \subseteq \{1, \dots, m\}$ let $w(I) = \sum_{i \in I} w(S_i)$. Let I be the solution returned by the algorithm, and let OPT be an optimal solution. Notice that

$$w(I) = \sum_{i=1}^n \text{price}(e_i)$$

since each set in I has its price distributed evenly among the new elements that it covered. We will show that $\text{price}(e_k) \leq w(\text{OPT})/(n - k + 1)$. If this is true, then

$$\begin{aligned} w(I) &= \sum_{i=1}^n \text{price}(e_i) \\ &\leq \sum_{i=1}^n \frac{w(\text{OPT})}{n - i + 1} \\ &= w(\text{OPT}) \left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{2} + 1 \right) \\ &= H(n) \cdot w(\text{OPT}) \end{aligned}$$

- **Claim:** For every $1 \leq k \leq n$, $\text{price}(e_i) \leq w(\text{OPT})/(n - k + 1)$

- Proof: Consider the iteration of the algorithm when e_k was covered for the first time. Let S_i be the set that was chosen by the algorithm, and let $C \subseteq U$ be the elements that were covered at the beginning of the iteration. Let OPT' denote the set of all $j \in \text{OPT}$ such that $S_j \setminus C \neq \emptyset$. Number the sets in OPT' S_1, \dots, S_ℓ in some order, and let $C_j = C \cup S_1 \cup \dots \cup S_j$. There exists $j \in \text{OPT}'$ such that $w(S_j)/|S_j \setminus C_j| \leq w(\text{OPT}')/|U \setminus C|$. To see this, assume on the contrary that $w(S_j)/|S_j \setminus C_j| > w(\text{OPT}')/|U \setminus C|$ for every $j \in \text{OPT}'$. Then

$$\begin{aligned}
w(\text{OPT}') &= \sum_{j=1}^{\ell} w(S_j) \\
&= \sum_{j=1}^{\ell} \frac{w(S_j)}{|S_j \setminus C_j|} \cdot |S_j \setminus C_j| \\
&> \sum_{j=1}^{\ell} \frac{w(\text{OPT}')}{|U \setminus C|} \cdot |S_j \setminus C_j| \\
&= \frac{w(\text{OPT}')}{|U \setminus C|} \cdot \sum_{j=1}^{\ell} |S_j \setminus C_j| \\
&= w(\text{OPT}')
\end{aligned}$$

which is a contradiction. So there exists $j \in \text{OPT}'$ such that

$$\begin{aligned}
\frac{w(S_j)}{|S_j \setminus C|} &\leq \frac{w(S_j)}{|S_j \setminus C_j|} \\
&\leq \frac{w(\text{OPT}')}{|U \setminus C|} \\
&\leq \frac{w(\text{OPT})}{|U \setminus C|}
\end{aligned}$$

Since our algorithm chose the most cost-efficient set S_i in the current round, we have

$$\begin{aligned}
\text{price}(e_k) &= \frac{w(S_i)}{|S_i \setminus C|} \\
&\leq \frac{w(S_j)}{|S_j \setminus C|} \\
&\leq \frac{w(\text{OPT})}{|U \setminus C|} \\
&\leq \frac{w(\text{OPT})}{n - k + 1}
\end{aligned}$$

The last inequality follows because there are at least $n - k + 1$ uncovered elements at the beginning of the iteration in which e_k is covered (since e_k, e_{k+1}, \dots, e_n are all uncovered).