Nonapproximable and Easily Approximable NP-Hard Optimization Problems

- In the previous lecture, we showed low-order polynomial time approximation algorithms for the two NP-Hard optimization problems of Bin Packing and Traveling Salesman (Restricted). We also showed that these approximation algorithms came with performance guarantees — in the sense that the solution returned can always be expected to be within a known (and reasonable) percentage of the optimum.
- In this lecture, we will examine the following two opposite ends of the approximability spectrum:
  - On the one hand, we will talk about NP-Hard optimization problems that appear to be nonapproximable. We consider an optimization problem nonapproximable when all attempted polynomial time approximation algorithms for the problem come with performance guarantees that are practically useless.
  - At the other end of the approximability spectrum, we will talk about the NP-Hard optimization problems that are considered easily approximable. An optimization problem is considered easily approximable if there exists a polynomial time approximation algorithm whose performance guarantee is such that you can expect the solution returned to be virtually the same as the optimum.

Examples of Nonapproximable Problems

The Graph Coloring Optimization Problem:

STATEMENT: Given a graph $G=(V,E)$, find a function $f: V \rightarrow [1,2,\ldots,k]$ such that $f(u) \neq f(v)$ whenever $\{u,v\} \in E$ and such that k is as small as possible.

Let A denote a polynomial time approximation algorithm for this problem. [A trivial such approximation algorithm would consist of simply assigning a different color to each vertex. In this case, $k = |V|$.] Let $A(G)$ denote the value of k returned by the algorithm. The best performance bound that

is currently known for any polynomial-time graph coloring algorithm is

$$\boxed{A(G) \leq \frac{c \cdot |V|}{\log |V|} OPT(G)}$$

Considering that as $|V|$ becomes larger and larger, $\boxed{\frac{|V|}{\log |V|}}$ rises almost as fast as $|V|$, this performance guarantee is almost as bad as that of the trivial algorithm. In other words, this is really not a usable performance guarantee. We may therefore claim that the Graph Coloring Optimization Problem is nonapproximable.

The Optimization Problem of Finding the Largest Independent Set (of Vertices) in a Graph:

As you will recall from Lectures 29 and 30, there must not exist an edge between any two vertices in an independent set.

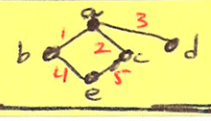
- No polynomial time approximation algorithms with any usable performance guarantee have been discovered for this problem. This is despite the fact the closely related problem of finding the smallest vertex cover is solvable in low-order polynomial time with a performance guarantee that says that the returned vertex cover will be of size no greater than twice that of the optimum. It follows from Lecture 30 that if V' is a smallest vertex cover if and only if $V-V'$ is the largest independent set for a graph $G=(V,E)$.

Recall that every edge of a graph must be represented by at least one of its two vertices in a vertex cover

- It seems paradoxical that we have two closely related optimization problems (one problem being the complement of the other, not in the decision-theoretic sense, but in the set-theoretic sense with regard to the set of all vertices in a graph), and yet while we have a good polynomial-time approximation algorithm for the vertex-cover minimization problem, no good approximation algorithm can be found for the independent-set maximization problem.

- The paradox disappears when you realize that even when you know that the ratio $\frac{x}{y}$ is bounded, we cannot make a similar statement about ratios like $\frac{c-x}{y}$ or $\frac{y}{c-x}$ for arbitrary c. To elaborate: The performance guarantee of the vertex-cover minimization algorithm in the box at right is given by

How to find the smallest vertex-cover in polynomial time with a reasonable performance guarantee: This algorithm is based on the idea of maximal matching in a graph. A matching in a graph $G=(V,E)$ is a set $E' \subseteq E$ of edges so that no two edges in E' share a common vertex. A matching is maximal if every remaining edge in $E-E'$ has a vertex in common with some edge in E'. A maximal matching E' can be constructed in polynomial time by scanning through all the edges in E and selecting a new edge for E' on the basis of no common vertices with the edges already in E'. A vertex cover can then be formed by taking the union of the vertices for the edges in E'. The size of this vertex cover will be $2|E'|$. Since all vertex covers for G must contain at least $|E'|$ vertices, the vertex cover returned by this algorithm is never more than twice as large as the minimum possible vertex cover.



$E' = \{1,5\}$
$VC = \{a,b,c,e\}$

$E' = \{3,4\}$
$VC = \{a,b,d,e\}$

$\frac{VC(I)}{OPT(I)} \leq 2$ where $VC(I)$ is the size of the vertex cover returned by the algorithm and $OPT(I)$ the actual size of the smallest vertex cover for instance I of the problem. The performance guarantee for the independent set maximization algorithm (that is based directly on the vertex cover minimization algorithm) would be the bound on the ratio $\frac{OPT(I)}{|V| - VC(I)}$. (For a maximization problem, we state our performance guarantee as a bound on $\frac{OPT(I)}{A(I)}$ where $A(I)$ is the answer returned by the algorithm.) Example: Consider a graph $G=(V,E)$ with $|V| = 10000$ and whose smallest vertex cover is of size 4990. The largest independent set for this graph will of size 5010. Now, our vertex cover minimization algorithm in the box could, in the worst case, return a vertex cover of size $2 \times 4990 = 9980$. That would result in an independent set of size $10000 - 9980 = 20$ as our approximation to the largest independent set. The performance ratio $\frac{OPT(I)}{A(I)}$ for this independent set calculation would be $\frac{5010}{20} \approx 250$!!

- The above discussion teaches us the following lesson:

> Even when two NP-Hard optimization problems are closely and directly related by a polynomial transformation, a performance-guaranteed polynomial-time approximation algorithm for one problem does NOT imply the existence of a performance-guaranteed approximation algorithm for the other.

Example of an Easily Approximable Problem

- That brings us to the other end of the approximability spectrum.
- NP-Hard optimization problems that can be solved by approximation algorithms with performance guarantees that ensure that the answer returned will be virtually the same as the optimum answer are usually the optimization versions of the $\boxed{\text{weak-sense}}$ NP-Complete decision problems. The optimization versions of weak-sense NP-Complete problems are referred to as weak-sense NP-Hard.
- The weak-sense NP-Hard problem we will consider is that of knapsack optimization:

The Knapsack Optimization Problem:

STATEMENT: Given a finite set $U = \{u_1, u_2, \ldots, u_n\}$, given a size function $s(u) \in \mathbb{Z}^+$, a value function $v(u) \in \mathbb{Z}^+$, and given a positive integer B that represents the knapsack's capacity, find a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leq B$ and such that $\sum_{u \in U'} v(u)$ is as large as possible.

For the algorithm at the bottom of this page the performance guarantee is proportional to $1+\frac{1}{k}$. It approaches 1 as k becomes large

- To understand the approximation algorithm that comes with a nearly perfect performance guarantee $\left(\frac{OPT(I)}{A(I)} \approx 1\right)$, you must first understand the following greedy algorithm whose performance guarantee is only $\frac{OPT(I)}{A(I)} \leq 2$. (Perhaps better thought of as $A(I) \geq \frac{OPT(I)}{2}$)

Did you know that the KNAPSACK decision problem discussed in Lecture 31 is NP-Complete only in the weak sense? As with the PARTITION problem in Lecture 35, the weak-sense NP-Completeness of KNAPSACK is established by providing a dynamic-programming based algorithm that yields the answer in time $O(nB)$ — as for PARTITION.

Greedy Algorithm for Knapsack Opt.:

① Order the set U by value density so that

$$\frac{v(u_1)}{s(u_1)} \geq \frac{v(u_2)}{s(u_2)} \geq \cdots\cdots \geq \frac{v(u_n)}{s(u_n)}$$

② Starting with U' empty, proceed sequentially through the ordered list shown above, each time adding the first unused u_i for which the available space is at least $s(u_i)$. In other words, at each step we select highest-value-density item that will fit in the knapsack (of size B).

③ When no more items can be placed in the knapsack, compare the value of the solution obtained with the solution consisting solely of there being only the maximum valued item in the knapsack. Take the better of the two.

- Let's now get back to the business of describing the approximation algorithm with nearly perfect performance guarantee. The main idea of this approximation algorithm is to examine all possible subsets of k or fewer objects from U. Each subset thus examined is expanded with the help of the Greedy Algorithm described above to fill the knapsack.

EXAMPLE:

Items in U:	u_1	u_2	u_3	u_4
size:	1	11	21	23
value:	11	21	31	33
value density:	11	1.9	1.47	1.43
Knapsack capacity $B = 45$				

0 = item not in knapsack
1 = item in knapsack

Optimum Solution by Exhaustive search:

u_1	u_2	u_3	u_4	Total Size	Total Value
0	0	0	1	23	33
0	0	1	0	21	37
0	0	1	1	⋮	⋮
0	0	0	0	⋮	⋮
1	0	1	1	45	75
⋮	⋮	⋮	⋮	⋮	⋮

15 rows

← Optimum $\{u_1, u_3, u_4\}$

Since U is already ordered by value density, the solution by the Greedy algorithm is seen directly as $\{u_1, u_2, u_3\}$ with value 63

▶ To illustrate the above algorithm, consider $k=2$ for the instance at left. So we need to construct all subsets of size 2 and less and extend each by the Greedy Algorithm:

# of k-subsets of a set of n items: $= \binom{n}{k} = \frac{n!}{k!(n-k)!}$

subset	size	value	Extension by Greedy	Extended size	Extended value
$\{u_1\}$	1	11	$+\{u_2, u_3\}$	33	63
$\{u_2\}$	11	21	$+\{u_1, u_3\}$	33	63
$\{u_3\}$	21	31	$+\{u_1, u_2\}$	33	63
$\{u_4\}$	23	33	$+\{u_1, u_2\}$	35	65
$\{u_1, u_2\}$	12	33	$+\{u_3\}$	33	64
$\{u_1, u_3\}$	22	42	$+\{u_2\}$	33	64
$\{u_1, u_4\}$	24	44	$+\{u_2\}$	35	65
$\{u_2, u_3\}$	33	52	$+\{u_1\}$	35	63
$\{u_2, u_4\}$	34	54	$+\{u_1\}$	45	75 *
$\{u_3, u_4\}$	44	64	$+\{u_1\}$	...	...