Motivation

If a problem you want to solve has been shown to be $\mathcal{NP}$-hard, your best bet is

- solve a more restricted version, or
- find an algorithm that computes a good approximation.

You may have gotten the impression that all $\mathcal{NP}$-complete problems are created equal.

- it is true that they are equivalent in the sense that they are equally hard to solve exactly
- but they are not equally hard to approximate.

Absolute and Relative Approximations

We shall aim for algorithms that are guaranteed to produce a result whose value R is within a certain proximity of the optimal value B.

The approximation is c-absolute if

$$B \geq R \geq B - c \quad \text{for maximization problems}$$
$$B \leq R \leq B + c \quad \text{for minimization problems}$$

The approximation is ϵ-relative if

$$B \geq R \geq B(1 - \epsilon) \quad \text{for maximization problems}$$
$$B \leq R \leq B(1 + \epsilon) \quad \text{for minimization problems}$$

Non-approximating Greedy Algorithms

Recall graph coloring: if (u, w) edge then u and w must have different color.

Problem: find the minimum number of colors needed.

Greedy Strategy: consider the nodes one by one

- assign the current node one of the colors used so far, if possible
- otherwise, use a new color

Now consider graph with

- nodes labeled $1..2n$
- edges connect all odd nodes to all even nodes, except no edges $(1, 2), (3, 4), \ldots$

There is a trivial 2-coloring. But the greedy strategy will assign 1,2 the same color which then cannot be reused, then 3,4 same color which then cannot be reused, etc, resulting in n colors being used.

Binary Knapsack

- find I to maximize $\sum_{i \in I} v_i$, given $\sum_{i \in I} w_i \leq W$
- greedy strategy G_0 picks most precious (value/weight ratio) items until no more space

This is non-approximating, since $R = 2$ while $B = N$ for
$w_1 = 1, v_1 = 2, w_2 = N, v_2 = N, W = N$

But we can get 0.5-relative (factor 2) by a simple trick:

1. use G_0 to produce I_0 with value R_0
2. return the best of I_0 and $\{M\}$ with V_M the highest v_i

Proof: assume items are ordered after preciousness, and that J be smallest with $W_J = w_1 + \ldots + w_J > W$. Observe that if the capacity had been W_J, G_0 would have yielded the optimal value B_J. Thus

$$\begin{aligned} R &= \max(R_0, V_M) \\ &\geq \max(v_1 + \ldots + v_{J-1}, v_J) \\ &\geq (v_1 + \ldots v_J)/2 = B_J/2 \geq B/2 \end{aligned}$$

Traveling Salesman

- We shall see that in the general case, it is $\mathcal{NP}$-hard to get a c-absolute or ϵ-relative approximation

But it is often the case that distances form metric:

$$d(x, y) \leq d(x, z) + d(z, y)$$

Then there is a 1-relative approximation:

1. construct (by Kruskal or Prim) minimum spanning tree T, with cost M. Since removing one edge from any Hamiltonian cycle is a spanning tree, $B \geq M$.
2. traverse T from root through leaves and back to root, thus visiting each edge twice so cost is $2M$.
3. Now make short-cuts when traveling from root to root, skipping nodes already visited. The resulting path has cost $R \leq 2M$, due to metric property.

We have found a Hamiltonian cycle, with cost $R \leq 2B$.

c-Absolute May Be Hard

Consider again the Traveling Salesman Problem

- assume that we in polynomial time can find a c-absolute approximation
- then we can also in polynomial time find a round trip that is exactly optimal (hence $\mathcal{P} = \mathcal{NP}$)

For given a distance map D, where we assume all distances are positive integers, and assume B is the minimal value of a round trip (Hamiltonian cycle). Then

1. construct a distance map D' from D, by multiplying all distances by $c + 1$. Thus $B' = B(c + 1)$.
2. call our purported approximative algorithm on D'; this returns a cycle Q with cost R' where

$$B(c + 1) = B' \leq R' \leq B' + c < (B + 1)(c + 1)$$

3. Return Q which wrt. D has cost $R = R'/(c + 1)$.

Thus $B \leq R < B + 1$ and hence $R = B$.

ϵ-Relative May Be Hard

- assume we in polynomial time can find ϵ-relative approximation to traveling salesman problem
- then we can also in polynomial time decide if a graph has a Hamiltonial cycle (and hence $\mathcal{P} = \mathcal{NP}$)

For given $G = (V, E)$, we

1. construct distance map d as follows:

$$\begin{array}{rcll} d(u,w) & = & 1 & \text{if } (u,w) \in E \\ d(u,w) & = & 2 + \lfloor n\epsilon \rfloor & \text{if } (u,w) \notin E \end{array}$$

   Observe this is in general not a metric.
2. Call our purported approximate algorithm on d, returning a cycle with cost R. With B the minimal cost, we have $B \leq R \leq B(1 + \epsilon)$.

Fact: G has Hamiltonial cycle iff $R \leq (1 + \epsilon)n$

- if G has Ham. cycle then $B = n$ so $R \leq (1 + \epsilon)n$.
- if G does not have a Hamiltonian cycle then

$$R \geq B \geq n + 1 + \lfloor n\epsilon \rfloor > n + \epsilon n = (1 + \epsilon)n.$$

MIN-CLUSTER and MAX-CUT

Even problems that appear dual may exhibit vastly different behavior. Consider MIN-CLUSTER/MAX-CUT:

- given complete graph where each edge has a cost
- we must split the nodes into 3 partitions (clusters)
- then some edges will be internal
- while the rest will be cross edges

This setting gives rise to two problems:

- MIN-CLUSTER: minimize the total cost of the internal edges
- MAX-CUT: maximize the total cost of the cross edges.

Clearly, an exactly solution to one will yield an exact solution to the other!

- but MAX-CUT can approximated efficiently
- while MIN-CLUSTER can not (unless $\mathcal{P} = \mathcal{NP}$).

MIN-CLUSTER: no efficient approximation

- assume that we in polynomial time can find an ϵ-relative approximation to MIN-CLUSTER.
- then 3-COL $\in \mathcal{P}$ and hence $\mathcal{P} = \mathcal{NP}$

For given $G = (V, E)$, we

1. construct costs c as follows:

$$\begin{array}{lcll} c(u,w) & = & 1 & \text{if } (u,w) \notin E \\ d(u,w) & = & n^2(1+\epsilon) & \text{if } (u,w) \in E \end{array}$$

2. Call our purported approximate algorithm on d, returning a partitioning with cost R. With B the minimum cost (sum of internal edges), we have $B \leq R \leq B(1+\epsilon)$.

Fact: G has 3-coloring iff $R < n^2(1+\epsilon)$.

- A 3-coloring induces partitioning where all internal edges have cost 1. Then $B < n^2$ so $R < n^2(1+\epsilon)$.
- if no 3-coloring exists one internal edge has cost $n^2(1+\epsilon)$, and hence $R \geq B \geq n^2(1+\epsilon)$.

MAX-CUT can be efficiently approximated

MAX-CUT has a $\frac{1}{3}$-relative approximation:

1. consider each node u in turn
   so as to place it in a cluster
2. consider the edges from u to the nodes previously considered
3. add u to the cluster that causes the sum of the internal edges to decrease least.

We infer that of the total cost C, at most one third will come from internal edges. With R the sum of cross edges in the resulting cluster, we thus have

$$R \geq \frac{2}{3}C \geq \frac{2}{3}B = (1 - \frac{1}{3})B$$

Binary Knapsack, Revisited

The simple greedy algorithm

- prioritizes after value/weight ratio
- can be arbitrarily imprecise
- but we can get a 0.5-relative approximation if we, whenever our selection is less valuable than the single most valuable item, take that item instead
- Can we get higher precision?

Idea: to get a $\frac{1}{k}$-relative approximation, we

1. generate all k-element subsets that fit;
2. for each such subset J, build a solution by running the simple greedy algorithm with J as initial value
3. pick the best such solution

Polynomial Approximation Scheme

Recall our algorithm feeds all possible k-element subsets as initial values to the simple greedy strategy, and picks the best such solution. With R the value of that solution, and B the optimal value, one can prove

$$R > \frac{Bk}{k+1} > \frac{B(k-1)}{k} = B(1 - \frac{1}{k})$$

and hence we have a $\frac{1}{k}$-relative approximation.

- When $k = 1$, we have the expected $R > B/2$.

But running time is in $\Theta(n^{k+1})$, so our high precision comes with a cost!

- This is a polynomial approximation scheme
- but we would rather like a fully polynomial approximation scheme.

A fully polynomial approximation scheme achieves $\frac{1}{k}$-relative approximation in time polynomial in n and in k.

Employing Dynamic Programming

We shall now construct a fully polynomial approximation scheme for the binary knapsack problem. First recall the dynamic programming algorithm for computing a table from which we can find an exact optimal solution:

> *the entry $V[i, w]$ denotes the maximum value we can get from items $1 \ldots i$ and weight limit w*

and is computed as follows:

- if $w = 0$ or $i = 0$ then 0
- else if $w < w_i$ then $V[i-1, w]$
- else $\max(V[i-1, w], V[i-1, w-w_i] + v_i)$.

Running Time is in $\Theta(nW)$.

- W may be exponential in size of input

Key to approximation: make the table smaller.

Twisting Dynamic Programming

To cut down the size of the dynamic programming table:

- divide numbers by big constant, ignoring remainders
- but dangerous to mess with weights, as rounding off may render a feasible solution infeasible, or vice versa
- rather mess with the values

We therefore reformulate dynamic programming so that it constructs a table indexed by values:

> *an entry $C[i, v]$ denotes the minimum weight needed to get at least value v from items $\{1..i\}$*

Then the optimal value can be found as the largest v such that $C[n, v] \leq W$. Each entry is computed as follows:

- if $v \leq 0$ then 0
- else if $i = 0$ then ∞
- else $\min(C[i-1, v], C[i-1, v - v_i] + w_i)$

This runs in time $O(nV)$, where V is an upper bound of the optimal solution.

A Fully Polynomial Approximation Scheme

Let I be optimal solution of the problem, with value B.

1. Use cheap greedy algorithm to find R_0 such that

$$B/2 \leq R_0 \leq B.$$

2. Split into two cases:

   $R_0 < 2nk$: Then just apply dynamic programming, creating a table $W[0..n, 0..V]$ to compute the solution exactly.
   As $2R_0 \geq B$, we can pick $V = 2R_0$, and hence achieve a running time in $O(nR_0) \subseteq O(n^2k)$.

   $R_0 \geq 2nk$: see next page.

Fully Polynomial Approximation, part II

When $R_0 \geq 2nk$, with $d = \lfloor \frac{R_0}{nk} \rfloor$ we let

$$v'_i = \lfloor \frac{v_i}{d} \rfloor \text{ for } i \in I$$

and hence $dv'_i \leq v_i < dv'_i + d$. We now apply dynamic programming on this reduced problem, giving an optimal solution I' with value B'.

Let R be the value of I' wrt. the original values. Then

$$\begin{aligned} R &= \sum_{i \in I'} v_i \geq d \sum_{i \in I'} v'_i \geq d \sum_{i \in I} v'_i \\ &> \sum_{i \in I} (v_i - d) \geq B - dn \\ &\geq B - \frac{R_0}{k} \geq B - \frac{B}{k} = B(1 - \frac{1}{k}). \end{aligned}$$

The algorithm runs in time $O(n\frac{R_0}{d}) \subseteq O(n^2 k)$ (as case 1)