Approximation algorithms

\( \rho \)-approximation algorithm.

- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instances of the problem.
- Guaranteed to find solution within ratio \( \rho \) of true optimum.

**Ex.** Given a graph \( G \), the greedy algorithms finds a **VERTEX-COVER** that uses \( \leq 2 \text{OPT}(G) \) vertices in \( O(m + n) \) time.

**Challenge.** Need to prove a solution’s value is close to optimum value, without even knowing what optimum value is!
Algorithm VC \( (G = (V,E)) \)

\[
S = \emptyset
\]

\[
\text{while} \quad E \neq \emptyset \quad \text{do}
\]

\[
\text{pick an edge} \quad e = (u,v) \in E
\]

\[
S = S \cup \{u,v\}
\]

\[
\text{remove from } G \text{ all edges touching } u \text{ or } v
\]

\[
\text{end while}
\]

\[
\text{return } S
\]

Claim 1: Algorithm returns a vertex cover \( S \), & runs in polytime.

Claim 2: Let \( S \) be the output of our VC algorithm on \( G \). Let \( S^* \) be a min-size vertex cover for \( G \).

Then \[
|S| \leq 2 \cdot |S^*| \quad \text{approx factor}
\]
Proof of Claim 2:

\[ S = \{ (1, 2), (3, 4) \} \]

Observe: \( S \) is a matching (set of disjoint edges of \( G \)).

Any vertex cover for \( G \) must spend at least one vertex per edge in the matching:

\[ |S^*| \geq |\text{matching}| = \frac{|S|}{2} \]

Conjecture: Can't get a polytime algo. for VC with approx. factor \( < 2 \) unless \( P = NP \).
Knapsack problem

Knapsack problem.

- Given \( n \) objects and a knapsack.
- Item \( i \) has value \( v_i > 0 \) and weighs \( w_i > 0 \). \( \text{we assume } w_i \leq W \text{ for each } i \)
- Knapsack has weight limit \( W \).
- Goal: fill knapsack so as to maximize total value.

Ex: \( \{ 3, 4 \} \) has value 40.

<table>
<thead>
<tr>
<th>item</th>
<th>value</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

original instance (W = 11)
Knapsack is NP-complete

**KNAPSACK.** Given a set $X$, weights $w_i \geq 0$, values $v_i \geq 0$, a weight limit $W$, and a target value $V$, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} w_i \leq W$$
$$\sum_{i \in S} v_i \geq V$$

**SUBSET-SUM.** Given a set $X$, values $u_i \geq 0$, and an integer $U$, is there a subset $S \subseteq X$ whose elements sum to exactly $U$?

**Theorem.** $\text{SUBSET-SUM} \leq_p \text{KNAPSACK}$.

**Pf.** Given instance $(u_1, \ldots, u_n, U)$ of $\text{SUBSET-SUM}$, create $\text{KNAPSACK}$ instance:

$$v_i = w_i = u_i \quad \sum_{i \in S} u_i \leq U$$
$$V = W = U \quad \sum_{i \in S} u_i \geq U$$
Knapsack problem: dynamic programming I

**Def.** \( OPT(i, w) = \max \text{ value subset of items } 1, \ldots, i \text{ with weight limit } w. \)

**Case 1.** \( OPT \) does not select item \( i \).
- \( OPT \) selects best of \( 1, \ldots, i-1 \) using up to weight limit \( w \).

**Case 2.** \( OPT \) selects item \( i \).
- New weight limit = \( w - w_i \).
- \( OPT \) selects best of \( 1, \ldots, i-1 \) using up to weight limit \( w - w_i \).

\[
OPT(i, w) = \begin{cases} 
0 & \text{if } i = 0 \\
OPT(i-1, w) & \text{if } w_i > w \\
\max \{ OPT(i-1, w), \; v_i + OPT(i-1, w - w_i) \} & \text{otherwise}
\end{cases}
\]

**Theorem.** Computes the optimal value in \( O(n W) \) time.
- Not polynomial in input size.
- Polynomial in input size if weights are small integers.
Knapsack problem: dynamic programming II

**Def.** \( \text{OPT}(i, v) = \min \text{ weight of a knapsack for which we can obtain a solution of value } \geq v \text{ using a subset of items } 1, \ldots, i. \)

**Note.** Optimal value is the largest value \( v \) such that \( \text{OPT}(i, v) \leq W. \)

**Case 1.** \( \text{OPT} \) does not select item \( i. \)
- \( \text{OPT} \) selects best of \( 1, \ldots, i-1 \) that achieves value \( v. \)

**Case 2.** \( \text{OPT} \) selects item \( i. \)
- Consumes weight \( w_i, \) need to achieve value \( v - v_i. \)
- \( \text{OPT} \) selects best of \( 1, \ldots, i-1 \) that achieves value \( v - v_i. \)

\[
\text{OPT}(i, v) = \begin{cases} 
0 & \text{if } v \leq 0 \\
\infty & \text{if } i = 0 \text{ and } v > 0 \\
\min \{ \text{OPT}(i-1, v), w_i + \text{OPT}(i-1, v - v_i) \} & \text{otherwise}
\end{cases}
\]
Knapsack problem: dynamic programming II

**Theorem.** Dynamic programming algorithm II computes the optimal value in $O(n^2 \cdot \nu_{\text{max}})$ time, where $\nu_{\text{max}}$ is the maximum of any value.

**Pf.**
- The optimal value $V^k \leq n \cdot \nu_{\text{max}}$.
- There is one subproblem for each item and for each value $\nu \leq V^k$.
- It takes $O(1)$ time per subproblem. \(\blacksquare\)

**Remark 1.** Not polynomial in input size!

**Remark 2.** Polynomial time if values are small integers.
Knapsack problem: polynomial-time approximation scheme

Intuition for approximation algorithm.
- Round all values up to lie in smaller range.
- Run dynamic programming algorithm II on rounded instance.
- Return optimal items in rounded instance.

<table>
<thead>
<tr>
<th>item</th>
<th>value</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>934221</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5956342</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>17810013</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>21217800</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>27343199</td>
<td>7</td>
</tr>
</tbody>
</table>

original instance (W = 11)

<table>
<thead>
<tr>
<th>item</th>
<th>value</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

rounded instance (W = 11)
Knapsack problem: polynomial-time approximation scheme

Round up all values:
- \( v_{\text{max}} \) = largest value in original instance.
- \( \varepsilon \) = precision parameter.
- \( \theta \) = scaling factor = \( \varepsilon v_{\text{max}} / n \).

\[
\bar{v}_i = \left\lfloor \frac{v_i}{\theta} \right\rfloor \theta, \quad \hat{v}_i = \left\lfloor \frac{v_i}{\theta} \right\rfloor
\]

Observation. Optimal solutions to problem with \( \bar{v} \) are equivalent to optimal solutions to problem with \( \hat{v} \).

Intuition. \( \bar{v} \) close to \( v \) so optimal solution using \( \bar{v} \) is nearly optimal; \( \hat{v} \) small and integral so dynamic programming algorithm II is fast.
Knapsack problem: polynomial-time approximation scheme

Round up all values: \( \bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta \)

\( \bar{V}_{\text{max}} = \left\lceil \frac{V_{\text{max}} \cdot \theta}{\varepsilon} \right\rceil \cdot \frac{\varepsilon \cdot V_{\text{max}}}{n} \)

\( \bar{V}_{\text{max}} = V_{\text{max}} \) if \( \frac{1}{\varepsilon} \in \mathbb{Z} \) (is integer)

**Theorem.** If \( S \) is solution found by rounding algorithm and \( S^* \) is any other feasible solution, then \( (1+\varepsilon) \sum_{i \in S} v_i \geq \sum_{i \in S^*} v_i \)

**Pf.** Let \( S^* \) be any feasible solution satisfying weight constraint.

\[
\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \bar{v}_i \leq \sum_{i \in S} \bar{v}_i \leq \sum_{i \in S} (v_i + \theta) \leq \sum_{i \in S} v_i + n\theta \leq (1+\varepsilon) \sum_{i \in S} v_i
\]

always round up

solve rounded instance optimally

never round up by more than \( \theta \)

DP alg can take \( v_{\text{max}} \)

\( n \theta = \varepsilon v_{\text{max}}, V_{\text{max}} \leq \sum_{i \in S} v_i \)
Knapsack problem: polynomial-time approximation scheme

**Theorem.** For any $\varepsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1 + \varepsilon)$ factor of the optimum in $O(n^3 / \varepsilon)$ time.

**Pf.**
- We have already proved the accuracy bound.
- Dynamic program II running time is $O(n^2 \hat{v}_{\text{max}})$, where

$$\hat{v}_{\text{max}} = \left\lceil \frac{v_{\text{max}}}{\theta} \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

**PTAS.** $(1 + \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.
- Produces arbitrarily high quality solution.
- Trades off accuracy for time.
- But such algorithms are unlikely to exist for certain problems...
Inapproximability

**MAX-3-SAT.** Given a 3-SAT instance \( \Phi \), find an assignment that satisfies the maximum number of clauses.

**Theorem.** [Karloff-Zwick 1997] There exists a \( \frac{7}{8} \)-approximation algorithm.

**Theorem.** [Håstad 2001] Unless \( P = NP \), there does not exist a \( \rho \)-approximation for any \( \rho > \frac{7}{8} \).

---

A 7/8-Approximation Algorithm for MAX 3SAT?

*Howard Karloff*

*Uri Zwick*

We describe a randomized approximation algorithm which takes as input an instance of MAX 3SAT and finds an assignment that satisfies at least \( 7/8 \) of the clauses. Our algorithm is based on a reduction to MAX 2-SAT. We prove that the algorithm works for any \( \rho \) and that the approximation ratio is \( 7/8 \). The algorithm is based on a reduction to MAX 2-SAT. We prove that the algorithm works for any \( \rho \) and that the approximation ratio is \( 7/8 \).

---

Some Optimal Inapproximability Results

JOHAN HÅSTAD

Royal Institute of Technology, Stockholm, Sweden

Abstract. We prove a series of results about the hardness of approximating various problems in NP under the assumption \( P \neq NP \). Our results are based on the assumption that certain problems are hard to approximate, and we use this assumption to prove the hardness of approximation for other problems. We also discuss the implications of our results for the complexity of certain problems.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

---

Review

**Greedy**

**BFS**

**DFS**

**Div & Conq**

**DP**

**Flows**

Unknown

(randomized, quantum) efficient algorithms

(exact, approximation)

Remarks:...
Remarks:

- The classification of algs into Greedy, DP, etc. is just for convenience. It's OK to create hybrid algorithms.

- NP-complete problems:
  a source of many natural problems that we don't know how to solve with efficient algorithms.
  A common belief is no such algs exist (i.e., that P ≠ NP), but the truth is we don't really know!

- Randomized (and quantum) polynomial-time algo extend our notion of efficient algs (from the usual: deterministic poly-time algs).

- Our ideal algo is best (poly-time).
- fast (poly time)
- correct on all inputs.

in practice:

all inputs

it would suffice to have algos:
fast & correct on "actual inputs" only

Challenge:
1. Formalize "actual inputs"
2. Design algos & prove they work on "actual inputs".

(Analyze heuristics used in practice)