Slides for a Course on the Analysis and Design of Algorithms

Chapter 10: The Effective Solution of $\mathcal{NP}$-Complete Problems

Stephen J. Hegner
Department of Computing Science
Umeå University
Sweden
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

10. The Effective Solution of $\mathcal{NP}$-Complete Problems

10.1 Pseudo Polynomial-Time Algorithms

10.1.1 The *real* complexity of the discrete knapsack problem

The setting: (Similar to 3.1.2, except that all parameters are now integers.)

- A knapsack with weight capacity M.
- n objects $\{\mathsf{obj}_1, \mathsf{obj}_2, \ldots, \mathsf{obj}_n\}$, each with a weight w_i and a value v_i.
- M, the w_i's, and the v_i's are all taken to be positive integers.

- Recall that in the dynamic programming solution presented in 4.3.3, a sequence of sets was constructed:

$$S_0,\ S_1,\ S_2,\ \ldots,\ S_{n-1}$$

- In this sequence:
  - (i) Each S_i has length bounded by M, because only the most profitable entry for each weight is retained.
  - (ii) Each S_i may be constructed in time proportional to the size of S_{i-1}.
- Thus, the algorithm has complexity $O(n \cdot M)$.

Question: Is this a polynomial-time algorithm?

Answer with another question: Polynomial in what?

- This illustrates clearly the need to be very careful about how the size of the input is measured.
- It only requires $\lceil \log(M) \rceil$ bits to represent M, and the algorithm is not polynomial in $n \cdot \log(M)$.
- Nevertheless, it is clear that to achieve "intractability," very large values for M must be involved.
- In many reasonable cases, the discrete knapsack problem is not really so intractable after all.
- Note that these ideas apply to both the decision problem and the optimization problem.

10.1.2 Pseudo polynomial-time algorithms

- Let $P = (I, \rho)$ be a language problem.
- Assume that the elements of I are represented using a reasonable encoding, as described in 9.5.2.
- Let $w \in I$.

(a) $\max(w)$ denotes the largest integer which occurs in the problem encoding w.

(b) A deterministic algorithm A (*i.e.*, a DTM) for P is *pseudo polynomial-time* if there is a polynomial p with the property that for any $w \in I$, A solves P in time $\bar{O}(p(\mathsf{Length}(w), \max(w))$.

- In the discrete knapsack problem, it is clear that $\max(w) \geq M$, so the dynamic programming algorithm is indeed pseudo polynomial-time.

10.1.3 Remarks on the pseudo polynomial-time approach

- Many $\mathcal{NP}$-complete problems do not involve numbers, other than as labels on variables.
- A key example is SAT.
- Thus, there is no meaningful way even to analyze algorithms for SAT along the theme of pseudo polynomial-time computations.
- To characterize those problems which do allow such analysis, the following definition is made.

(a) The decision problem $P = (I, \rho)$ is a *number problem* if

  (i) It uses numbers as computational values in the problem description, and not just as labels;

  (ii) There is no polynomial p such that, for all $w \in I$, $\max(w) \leq p(\mathsf{Length}(w))$.

- The motivation behind (ii) above is that the numbers which occur in problem instances may be arbitrarily large.
- Even many number problems, such as travelling salesman, do not admit pseudo polynomial-time solutions.
- A small theory can be developed along these lines, but the details are not presented here.

10.2 Approximation Algorithms

10.2.1 The setting: a return to optimization problems

- In the study of approximation algorithms, the context returns to that of optimization problems.
- The idea is to seek algorithms which, while not always producing optimal solutions, find ones which are not more than a specified amount from optimal.
- It is often possible to find such algorithms which run in deterministic polynomial time, even in the case that the original (corresponding decision) problem is $\mathcal{NP}$-complete.

10.2.2 Multisolution problems and their properties In this definition, let Σ be a finite alphabet.

(a) A *multisolution problem* (over Σ) is a quadruple $P = (I, S, \rho, r)$ in which

(i) $I \subseteq \Sigma^*$ is a set of *problem instances*.

(ii) $S \subseteq \Sigma^*$ is a set of *solution instances*.

(iii) $\rho : I \to \mathbf{2}^S$ is the *solution relation*.

(iv) $r : S \to \mathbb{N}$ is the *solution ranking function*.

(b) If $w \in I$ and $s \in \rho(w)$, then s is called a *feasible solution* for w.

(c) An *optimal solution* for $w \in I$ with respect to r is a feasible solution $s \in S$ with the further property that:

$$r(s) = \max(\{r(t) \mid t \in \rho(w)\})$$

The value $r(s)$ so identified is written $\mathsf{Opt}_r(w)$.

10.2.3 Example context – the discrete knapsack optimization problem

- The discrete knapsack optimization problem, as described in 3.1.2, will be used as a running example.
- The following (obvious) conventions will always be followed.
  (i) The weights and values of the objects, as well as the capacity of the knapsack, will always be taken to be positive integers.
  (ii) The set S of feasible solutions consists exactly of those subsets of objects whose total weight does not exceed the capacity of the knapsack.
  (iii) The ranking function $r : S \to \mathbb{N}$ assigns to each solution s the sum of the values of its objects.

10.2.4 Algorithms and absolute approximation Let $P = (I, S, \rho, r)$ be a multisolution problem over alphabet Σ. P.

(a) The deterministic Turing machine $M = (Q, \Sigma, \Gamma, \#, \delta, q_0, F)$ *solves* P if, for each $w \in I$,

$$f_M(w) \in \rho(w)$$

(b) For $\varepsilon \in \mathbb{R}^{>0}$, M is an ε-*absolute approximation algorithm* for P with respect to r if, for all $w \in I$,

$$|\mathsf{Opt}_r(w) - r(f_M(w))| \leq \varepsilon$$

(c) If M computes f_M in deterministic polynomial time, then it is said to be an ε-*absolute polynomial-time approximation algorithm* for P.

10.2.5 Theorem *For no $\varepsilon \in \mathbb{R}^{>0}$ can there be an ε-absolute polynomial-time approximation algorithm for the discrete knapsack optimization problem unless $\mathcal{P} = \mathcal{NP}$.*

PROOF: Suppose that M is such an approximation algorithm for $\varepsilon > 0$, and let w be an instance of the discrete knapsack problem with the property that

$$|\mathsf{Opt}_r(w) - r(f_M(w))| \leq \varepsilon$$

Now create a new instance w' from w by multiplying the value of each object by $\lceil \varepsilon + 1 \rceil$. That is, if the original value of obj_i was v_i, then the new value is $\lceil \varepsilon + 1 \rceil \cdot \mathsf{v}_i$. It is easy to see that w' and w have exactly the same feasible solutions, with the total value of each solution for w' $\lceil \varepsilon + 1 \rceil$ times as large than the value of the corresponding solution of w. In particular, the value of each solution must be an integer multiple of $\lceil \varepsilon + 1 \rceil$. Thus, the only way that

$$|\mathsf{Opt}_r(w') - r(f_M(w'))| \leq \varepsilon$$

can hold is if

$$|\mathsf{Opt}_r(w') - r(f_M(w'))| = 0$$

Now, it is obvious that if the discrete knapsack optimization problem could be solved exactly, then so too could the corresponding decision problem. Hence, this condition can only be met in the case that $\mathcal{P} = \mathcal{NP}$. □

10.2.6 Relative approximation Let $P = (I, S, \rho, r)$ be a multisolution problem over alphabet Σ. and let $M = (Q, \Sigma, \Gamma, \#, \delta, q_0, F)$ be a DTM which solves P.

(a) For $\varepsilon \in \mathbb{R}^{>0}$, M is said to be a ε-*relative approximation algorithm* for P with respect to r if, for all $w \in I$ for which $\mathsf{Opt}_r(w) > 0$,

$$\frac{|\mathsf{Opt}_r(w) - r(f_M(w))|}{\mathsf{Opt}_r(w)} \leq \varepsilon$$

(b) If M computes f_M in deterministic polynomial time, then it is said to be an ε-*relative polynomial-time approximation algorithm* for P.

10.2.7 An approximation for the discrete knapsack problem

- The input to this algorithm is a discrete knapsack problem with n objects, as described in 3.1.2 and 10.2.3.
- Specifically, the context and notation is as follows:
  - A knapsack with weight capacity M.
  - n objects $\{\mathsf{obj}_1, \mathsf{obj}_2, \ldots, \mathsf{obj}_n\}$, each with a weight w_i and a value v_i.
  - M, the w_i's, and the v_i's are all taken to be positive integers.
- Additionally, an integer parameter $k > 0$, which is the *order* of the algorithm, is supplied.
- The general idea is that, the larger k is, the better the approximation will be.
- Shown on the next page is a high-level sketch of the algorithm.
- The complexity analysis which follows (10.2.8) will be based upon the times in a high-level language, rather than those of a Turing machine, but the principle that the difference will be limited to a polynomial transformation should be kept in mind.

- $\mathcal{S}_{k,n}$ denotes the set of all subsets of $\{1,2,\ldots,n\}$ which contain no more than k elements.
- The function $\mathsf{GrKnap}_{p/w}(B,W)$ gives the set of objects selected when a greedy-style algorithm is applied to just the objects in $\{\mathsf{obj}_i \mid i \in B\}$, with a reduced knapsack capacity of W. The objects are considered in order of non-increasing profit per unit weight.
- The solution is represented as a subset of $\{1,2,\ldots,n\}$, as opposed to a vector $x \in \{0,1\}^n$.

$best\_profit \leftarrow 0$;
$best\_solution \leftarrow \emptyset$;
foreach $A \in \mathcal{S}_{k,n}$ **do**
  if $(\sum_{i\in A} \mathsf{w}_i \leq M)$
    then $\langle$ $\mathsf{kProfit}(A) \leftarrow \sum_{i\in A} \mathsf{v}_i$;
      $tent\_solution \leftarrow A \cup \mathsf{GrKnap}_{p/w}(\{1,\ldots,n\} \setminus A, M - \sum_{i\in A} \mathsf{w}_i)$;
      if $(\sum_{i\in tent\_solution[i]} \mathsf{v}_i > best\_profit)$
        then $\langle$ $best\_solution \leftarrow tent\_solution$;
          $best\_profit \leftarrow \sum_{i\in A} \mathsf{v}_i \cdot best\_solution[i]$;
        $\rangle$
    $\rangle$

10.2.8 Theorem *The algorithm presented in 10.2.7 is a* $(1/(k+1))$*-relative polynomial-time approximation algorithm for the discrete knapsack optimization problem which runs in worst case time* $O(n^{k+1})$.

PROOF:

- The complete context of 10.2.7 is assumed.
- Let $\mathsf{S}_{\mathsf{opt}} \subseteq \{1,2,\ldots,n\}$ define an optimal solution, with $b \stackrel{\text{def}}{=} \mathsf{Card}(\mathsf{S}_{\mathsf{opt}})$.
- Assume that $k < b$; otherwise, the algorithm finds an optimal solution directly.
- Let S_k denote the subset of $\mathsf{S}_{\mathsf{opt}}$ which indexes the k most valuable objects.
- Order the indices of the jobs in $\mathsf{S}_{\mathsf{opt}}$ as $\langle \alpha_1, \alpha_2, \ldots, \alpha_k, \alpha_{k+1}, \ldots, \alpha_b \rangle$ according to the following scheme.

$$\underbrace{\mathsf{v}_{\alpha_1} \geq \mathsf{v}_{\alpha_2} \geq \ldots \geq \mathsf{v}_{\alpha_k}}_{\mathsf{S}_k} \qquad \underbrace{\frac{\mathsf{v}_{\alpha_{k+1}}}{\mathsf{w}_{\alpha_{k+1}}} \geq \frac{\mathsf{v}_{\alpha_{k+1}}}{\mathsf{w}_{\alpha_{k+2}}} \geq \ldots \geq \frac{\mathsf{v}_{\alpha_b}}{\mathsf{w}_{\alpha_b}}}_{\text{other objects of } \mathsf{S}_{\mathsf{opt}}}$$

- Let $\mathsf{S}_{\mathsf{alg}}$ denote the solution found by the algorithm with the set $\mathsf{S}_k \in \mathcal{S}_{k,n}$ the choice for A in the foreach loop.
- Thus, $\mathsf{S}_{\mathsf{gr}} \stackrel{\text{def}}{=} \mathsf{S}_{\mathsf{alg}} \setminus \mathsf{S}_k$ is the set of objects which the algorithm adds using the greedy-style strategy.
- Define

$$\mathsf{Val}(\mathsf{S}_k) = \sum_{x \in \mathsf{S}_k} \mathsf{v}_x \qquad \mathsf{Val}(\mathsf{S}_{\mathsf{opt}}) = \sum_{x \in \mathsf{S}_{\mathsf{opt}}} \mathsf{v}_x$$

$$\mathsf{Val}(\mathsf{S}_{\mathsf{alg}}) = \sum_{x \in \mathsf{S}_{\mathsf{alg}}} \mathsf{v}_x \qquad \mathsf{Val}(\mathsf{S}_{\mathsf{gr}}) = \sum_{x \in \mathsf{S}_{\mathsf{gr}}} \mathsf{v}_x$$

- If no rejection occur during the greedy-style computation, the entire solution is optimal.
- Otherwise, let α_m be the index of the first object in $\mathsf{S}_{\mathsf{opt}}$ which is rejected by the greedy subalgorithm.
- Let T denote the amount of capacity remaining in the knapsack at the time of this rejection. Note that

  $$T < M - \sum_{i=1}^{m-1} \mathsf{w}_{\alpha_i}$$

  but that equality can not hold; otherwise α_m would be included by the greedy procedure.
- Let Δ denote the amount of space used by the objects which were not in $\mathsf{S}_{\mathsf{opt}}$ but which were selected by the greedy subalgorithm at the point at which obj_{α_m} was rejected. Thus,

  $$\Delta = (M - \sum_{i=1}^{m-1} \mathsf{w}_{\alpha_i}) - T$$

- It must be the case that

  $$\sum_{i=k+1}^{m-1} \mathsf{v}_{\alpha_i} + \left(\frac{\mathsf{v}_{\alpha_m}}{\mathsf{w}_{\alpha_m}}\right) \cdot \Delta \;\leq\; \mathsf{Val}(\mathsf{S}_{\mathsf{gr}})$$

  since any object obj_ℓ selected by the greedy subalgorithm before obj_{α_m} must have $\mathsf{v}_\ell/\mathsf{w}_\ell \geq \mathsf{v}_m/\mathsf{w}_m$.
- Also, the following must hold

  $$\sum_{i=m}^{b} \mathsf{v}_{\alpha_i} \leq \left(\frac{\mathsf{v}_{\alpha_m}}{\mathsf{w}_{\alpha_m}}\right) \cdot (M - \sum_{i=1}^{m-1} \mathsf{w}_{\alpha_i}) = \left(\frac{\mathsf{v}_{\alpha_m}}{\mathsf{w}_{\alpha_m}}\right) \cdot (\Delta + T)$$

  since the objects $\{\mathsf{obj}_{\alpha_m}, \ldots, \mathsf{obj}_{\alpha_b}\}$ can fill at most the rest of the knapsack, and can have p/w no greater than $\mathsf{v}_{\alpha_m}/\mathsf{w}_{\alpha_m}$.

- Now for the optimal profit:

$$\begin{aligned}
&\mathsf{Val}(\mathsf{S}_{\mathsf{opt}}) \\
&= \mathsf{Val}(\mathsf{S}_k) + \sum_{i=k+1}^{b} \mathsf{v}_{\alpha_i} \\
&= \mathsf{Val}(\mathsf{S}_k) + \sum_{i=k+1}^{m-1} \mathsf{v}_{\alpha_i} + \sum_{i=m}^{b} \mathsf{v}_{\alpha_i} \\
&\leq \mathsf{Val}(\mathsf{S}_k) + (\mathsf{Val}(\mathsf{S}_{\mathsf{gr}}) - \left(\frac{\mathsf{v}_{\alpha_m}}{\mathsf{w}_{\alpha_m}}\right) \cdot \Delta) + \left(\frac{\mathsf{v}_{\alpha_m}}{\mathsf{w}_{\alpha_m}}\right) \cdot (M - \sum_{i=1}^{m-1} \mathsf{w}_{\alpha_i}) \\
&= \mathsf{Val}(\mathsf{S}_k) + \mathsf{Val}(\mathsf{S}_{\mathsf{gr}}) + \left(\frac{\mathsf{v}_{\alpha_m}}{\mathsf{w}_{\alpha_m}}\right) \cdot T \\
&< \mathsf{Val}(\mathsf{S}_k) + \mathsf{Val}(\mathsf{S}_{\mathsf{gr}}) + \mathsf{v}_{\alpha_m} \qquad (\text{since } T/\mathsf{w}_{\alpha_m} < 1) \\
&= \mathsf{Val}(\mathsf{S}_{\mathsf{alg}}) + \mathsf{v}_{\alpha_m}
\end{aligned}$$

- Since

$$\mathsf{Val}(\mathsf{S}_k) + \mathsf{v}_{\alpha_m} \leq \mathsf{Val}(\mathsf{S}_{\mathsf{opt}})$$

  the average profit of the set $\{\mathsf{obj}_i \mid i \in \mathsf{S}_k\} \cup \mathsf{obj}_{\alpha_m}$ cannot exceed $\mathsf{Val}(\mathsf{S}_{\mathsf{opt}})/(k+1)$.

- Since S_k indexes the k most profitable objects of $\mathsf{S}_{\mathsf{opt}}$, obj_{α_m} must be the least profitable of those indexed by $\{\mathsf{obj}_i \mid i \in \mathsf{S}_k\} \cup \mathsf{obj}_{\alpha_m}$; *i.e.*,

$$\mathsf{v}_{\alpha_m} \leq \frac{\mathsf{Val}(\mathsf{S}_{\mathsf{opt}})}{(k+1)}$$

- Hence

$$\begin{aligned}
\left|\frac{\mathsf{Val}(\mathsf{S}_{\mathsf{opt}}) - \mathsf{Val}(\mathsf{S}_{\mathsf{alg}})}{\mathsf{Val}(\mathsf{S}_{\mathsf{opt}})}\right| &< \left|\frac{\mathsf{Val}(\mathsf{S}_{\mathsf{alg}}) + \mathsf{v}_{\alpha_m} - \mathsf{Val}(\mathsf{S}_{\mathsf{alg}})}{\mathsf{Val}(\mathsf{S}_{\mathsf{opt}})}\right| \\
&= \frac{\mathsf{v}_{\alpha_m}}{\mathsf{Val}(\mathsf{S}_{\mathsf{opt}})} \leq \frac{1}{(k+1)}
\end{aligned}$$

- Thus, the algorithm satisfies the $(k+1)$-relative approximation property
- It only remains to show that it runs in time $O(n^{k+1})$.
- It suffices to note that

$$\sum_{i=0}^{k} \binom{n}{i} \leq \sum_{i=0}^{k} n^i \in O(n^k)$$

- Thus, there are $O(n^k)$ subsets of $\{1,2,\ldots,n\}$ to consider.
- For each subset, the algorithm runs in linear time.
- The only other point of complexity is that for the greedy-style addition, the elements must be sorted. However, this only need be done once for the entire problem.
- Thus, for any $k \geq 1$, the algorithm runs in time $O(n^{k+1})$.

□

10.2.9 Fact *The bound $O(n^{k+1})$ in 10.2.8 is tight, so that the complexity is in fact $\Theta(n^{k+1})$.*

PROOF: Consult the text of Horowitz, Sahni, and Rajasekaran. □

10.3 Fully Polynomial-Time Approximation Methods

10.3.1 Approximation schemes Let $P = (I, S, \rho, r)$ be a multisolution problem.

(a) An *approximation scheme* for P is an algorithm M which accepts as input

(i) an instance $w \in I$; and

(ii) a real number $\varepsilon > 0$

and produces as output a solution $M(w, \varepsilon)$ which satisfies

$$\frac{|\mathsf{Opt}_r(w) - r(M(w,\varepsilon))|}{\mathsf{Opt}_r(w)} \leq \varepsilon$$

whenever $\mathsf{Opt}_r(w) > 0$.

(b) The approximation scheme M is *fully polynomial* if there is a polynomial p of two variables such that given any $w \in I$ and $\varepsilon > 0$, M runs in time $O(p(\mathsf{Length}(w)), 1/\varepsilon)$.

10.3.2 Observation The algorithm of 10.2.7 and 10.2.8 is not fully polynomial time, since $n^{1/\varepsilon}$ is not a polynomial in n and $1/\varepsilon$.

10.3.3 Knapsack problems recast

(a) An *n-element knapsack problem* is a triple

$$P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$$

in which

(i) The function

$$\mathsf{v}_{(-)} : \{1, 2, \ldots, n\} \to \mathbb{R}^{\geq 0}$$

assigns to each $i \in \{1, 2, \ldots, n\}$ a value v_i for the i^{th} object.

(ii) The function

$$\mathsf{w}_{(-)} : \{1, 2, \ldots, n\} \to \mathbb{N}$$

assigns to each $i \in \{1, 2, \ldots, n\}$ a weight w_i for the i^{th} object.

(iii) $M \in \mathbb{N}$ is the *capacity* of the knapsack, with $M \geq \mathsf{w}_i$ for $1 \leq i \leq n$.

(b) A *feasible solution* is any vector $(x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$ for which

$$\sum_{i=1}^{n} x_i \cdot \mathsf{w}_i \leq M$$

(c) An *optimal solution* is a feasible solution $(x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$ for which

$$\sum_{i=1}^{n} x_i \cdot \mathsf{v}_i$$

is a maximum over all feasible solutions. $\mathsf{Opt}(P)$ denotes this sum for an optimal solution.

(d) If $\mathsf{v}_i \in \mathbb{N}$ for each i, then P is called an *integer knapsack problem*.

10.3.4 Partial solutions to knapsack problems

Let $n \in \mathbb{N}$, and let $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$ be an n-element knapsack problem.

(a) For $k \leq n$, a *k-partial solution* for P is any k-tuple

$$(x_1, x_2, \ldots, x_k) \in \{0,1\}^k$$

with the property that there is a feasible solution $(y_1, y_2, \ldots, y_n)$ satisfying

$$x_i = y_i \text{ for } 1 \leq i \leq k$$

(b) In the above, $(y_1, y_2, \ldots, y_n)$ is called a *completion* of $(x_1, x_2, \ldots, x_k)$ with respect to P.

(c) If $X = (x_1, x_2, \ldots, x_k)$ and $Z = (z_1, z_2, \ldots, z_k)$ are k-partial solutions of P, X is said to *dominate* Z if there is some completion $(y_1, y_2, \ldots, y_n)$ of X with the property that

$$\sum_{i=1}^{n} y_i \cdot \mathsf{v}_i \geq \sum_{i=1}^{n} w_i \cdot \mathsf{v}_i$$

for every completion $(w_1, w_2, \ldots, w_n)$ of Z.

10.3.5 A common framework for the three techniques

- Three techniques, *rounding*, *interval partitioning*, and *separation* will be described.
- Variations of the dynamic-programming solution described in 4.3 will be used.
- Either the problem instance (in rounding), or the method of purging (in interval partitioning and separation) will be modified to yield approximate solutions under conditions of lowered complexity.

10.4 Rounding

10.4.1 Rounding – the basic idea

- In the rounding technique, a new instance P'' of the knapsack problem is created from the original instance P.
- The algorithm of 4.3 is not modified at all; only the instance is.
- In P'', the the values of the objects are rounded in such a way that the number of possible entries in each S_i is reduced substantially, while keeping the error within specified bounds.

10.4.2 Lemma *Let* $\varepsilon > 0$, $n \in \mathbb{N}$, *and let*

$$P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M) \qquad P' = (\mathsf{v}'_{(-)}, \mathsf{w}'_{(-)}, M')$$

be two instances of the n-element knapsack problem, with the further properties that $M = M'$ *and* $\mathsf{w}_i = \mathsf{w}'_i$ *for* $1 \leq i \leq n$. *If*

$$\sum_{i=1}^{n} |\mathsf{v}_i - \mathsf{v}'_i| \leq \varepsilon \cdot \mathsf{Opt}(P)$$

then

$$\frac{|\mathsf{Opt}(P) - \mathsf{Opt}(P')|}{\mathsf{Opt}(P)} \leq \varepsilon$$

PROOF: This is immediate, since

$$|\mathsf{Opt}(P) - \mathsf{Opt}(P')| \leq \sum_{i=1}^{n} |\mathsf{v}_i - \mathsf{v}'_i|$$

□

10.4.3 Acceptable lower bounds and the modified instance Let $\varepsilon > 0$, $n \in \mathbb{N}$, and let $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$ be an instance of the n-element knapsack problem.

(a) An *acceptable lower bound* for P is any number $\mathsf{LB}(P) \in \mathbb{N}$ which satisfies the following conditions:

(i) $\mathsf{LB}(P) \leq \mathsf{Opt}(P)$.

(ii) $\mathsf{LB}(P) \geq \mathsf{v}_i$ for $1 \leq i \leq n$.

- As an example, $\mathsf{LB}(P)$ may be taken to be a value computed by a greedy-style method.

(b) Define $P^{(\mathsf{LB},\varepsilon)} = (\mathsf{v}_{(-)}^{(\mathsf{LB},\varepsilon)}, \mathsf{w}_{(-)}, M)$ to be the n-element knapsack problem which has the same weights and knapsack capacity as P, but which has object values as defined below.

$$\mathsf{v}_i^{(\mathsf{LB},\varepsilon)} \stackrel{\text{def}}{=} \mathsf{v}_i - (\mathsf{v}_i) \bmod ((\mathsf{LB}(P) \cdot \varepsilon)/n)$$

- This is the largest multiple of $(\mathsf{LB}(P) \cdot \varepsilon)/n$ which is no larger than v_i.
- Note that the $\mathsf{v}_i^{(\mathsf{LB},\varepsilon)}$'s need not be integers, even if the v_i's are.
- This is not a serious problem, and, in any case, an equivalent instance with integer values will be constructed shortly.

10.4.4 Lemma *With the definitions as in 10.4.3,*

$$\sum_{i=1}^{n} |\mathsf{v}_i - \mathsf{v}_i^{(\mathsf{LB},\varepsilon)}| < \varepsilon \cdot \mathsf{Opt}(P)$$

PROOF:

$$|\mathsf{v}_i - \mathsf{v}_i^{(\mathsf{LB},\varepsilon)}| < \frac{\mathsf{LB}(P) \cdot \varepsilon}{n}$$

just by definition. Hence

$$\sum_{i=1}^{n} |\mathsf{v}_i - \mathsf{v}_i^{(\mathsf{LB},\varepsilon)}| < \left(\frac{\mathsf{LB}(P) \cdot \varepsilon}{n}\right) \cdot n = \mathsf{LB}(P) \cdot \varepsilon \leq \mathsf{Opt}(P) \cdot \varepsilon$$

□

10.4.5 Theorem

- *Let A be any algorithm which solves the discrete knapsack problem optimally.*
- *Let* $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$ *be any instance of the n-element knapsack problem.*
- *Let* $\varepsilon > 0$*, and let* $\mathsf{LB}(P)$ *be an acceptable lower bound for P.*
- *Let* $P^{(\mathsf{LB},\varepsilon)} = (\mathsf{v}_{(-)}^{(\mathsf{LB},\varepsilon)}, \mathsf{w}_{(-)}, M)$ *be the associated problem, as defined in 10.4.3(b).*
- *Then the algorithm which first converts P to* $P^{(\mathsf{LB},\varepsilon)}$ *and then applies A to this resulting instance is an* ε*-relative approximation algorithm.*

PROOF: Just combine 10.4.2 and 10.4.4. □

10.4.6 Conversion of $P^{(\mathsf{LB},\varepsilon)}$ to a problem with integer object values

- To use 10.4.5 as a tool to construct an ε-relative approximation algorithm which runs in polynomial time, is is first necessary to convert $P^{(\mathsf{LB},\varepsilon)}$ to an equivalent problem with integer object values.
- This is easy; just multiply each value $\mathsf{v}_i^{(\mathsf{LB},\varepsilon)}$ by $n/(\mathsf{LB}(P)\cdot\varepsilon)$.

(a) More specifically, define

$$\begin{aligned}\mathsf{v}_i^{(\overline{\mathsf{LB},\varepsilon})} &\stackrel{\text{def}}{=} \mathsf{v}_i^{(\mathsf{LB},\varepsilon)} \cdot \left(\frac{n}{\mathsf{LB}(P)\cdot\varepsilon}\right) \\ &= \left\lfloor \frac{\mathsf{v}_i \cdot n}{\mathsf{LB}(P)\cdot\varepsilon} \right\rfloor\end{aligned}$$

- The last equality holds because $\mathsf{v}_i^{(\mathsf{LB}(P),\varepsilon)}$ is the largest multiple of $(\mathsf{LB}(P)\cdot\varepsilon)/n$ which is no greater than v_i.

(b) Define $P^{(\overline{\mathsf{LB},\varepsilon})} = (\mathsf{v}_{(-)}^{(\overline{\mathsf{LB},\varepsilon})}, \mathsf{w}_{(-)}, M)$ to be the n-element knapsack problem using the above definition for values.

10.4.7 Lemma *Let $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$ be any instance of the n-element knapsack problem. Then, for any acceptable lower bound LB for P and any $\varepsilon > 0$, $P^{(\mathsf{LB},\varepsilon)}$ and $P^{(\overline{\mathsf{LB},\varepsilon})}$ have the same feasible and optimal solutions.*

PROOF: This is immediate, since only the profits have been scaled, and all by the same amount. □

10.4.8 Lemma *Let $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$ be any n-element knapsack problem with integer values. When the dynamic programming algorithm of 4.3 is applied to the modified instance $P^{(\overline{\mathsf{LB},\varepsilon})} = (\mathsf{v}_{(-)}^{(\overline{\mathsf{LB},\varepsilon})}, \mathsf{w}_{(-)}, M)$, the following constraint holds on the size of the intermediate sets of the form S_i which are constructed.*

$$\sum_{i=1}^{n-1} \mathsf{Card}(S_i) \leq n + \left(\frac{n\cdot(n-1)}{2}\right) \cdot \left\lfloor \frac{n}{\varepsilon} \right\rfloor \in O(n^3/\varepsilon)$$

PROOF:

- Since $\mathsf{v}_i \leq \mathsf{LB}(P)$ for $1 \leq i \leq n$,

$$\mathsf{v}_i^{(\overline{\mathsf{LB},\varepsilon})} \leq \left\lfloor \frac{\mathsf{LB}(P)\cdot n}{\mathsf{LB}(P)\cdot\varepsilon} \right\rfloor = \left\lfloor \frac{n}{\varepsilon} \right\rfloor$$

- Hence

$$\mathsf{Card}(S_i) \leq 1 + \sum_{j=1}^{i} \mathsf{v}_i^{(\overline{\mathsf{LB},\varepsilon})} \leq 1 + i \cdot \left\lfloor \frac{n}{\varepsilon} \right\rfloor$$

- The term "1" is added to account for the entry $\binom{0}{0}$.

- Thus

$$\sum_{i=0}^{n-1} \mathsf{Card}(S_i) \leq \sum_{i=0}^{n-1} \left(1 + i \cdot \left\lfloor \frac{n}{\varepsilon} \right\rfloor\right) = n + \left(\frac{n\cdot(n-1)}{2}\right) \cdot \left\lfloor \frac{n}{\varepsilon} \right\rfloor \in O(n^3/\varepsilon)$$

□

- The following is a restatement of 10.4.5, modified to take into account the use of $P^{(\overline{\mathsf{LB},\varepsilon})}$ and the complexity result of 10.4.8.

10.4.9 Theorem – final description of rounding

- *Let A be any algorithm which solves the discrete knapsack problem optimally.*
- *Let* $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$ *be any instance of the n-element knapsack problem.*
- *Let* $\varepsilon > 0$*, and let* $\mathsf{LB}(P)$ *be an acceptable lower bound for P.*
- *Let* $P^{(\overline{\mathsf{LB},\varepsilon})} = (\mathsf{v}^{(\overline{\mathsf{LB},\varepsilon})}_{(-)}, \mathsf{w}_{(-)}, M)$ *be the associated problem, as defined in 10.4.6.*
- *Then the algorithm which first converts P to* $P^{(\overline{\mathsf{LB},\varepsilon})}$ *and then applies A to this resulting instance is an* ε*-relative approximation algorithm which runs in time* $O(n^3/\varepsilon)$*.* □

10.5 Interval Partitioning and Separation

10.5.1 The basic idea behind interval partitioning and separation

- These techniques differ from rounding in that the problem instance is not modified.
- Rather, when two pairs $\binom{p_i}{w_i}, \binom{p_j}{w_j} \in S_i$ are close together, only one is retained.
- In this way, the size of each S_i's is limited.
- Interval partitioning differs from separation in the manner in which the choices of which elements to discard is made.

10.5.2 The interval partitioning algorithm

Input:

- An n-element integer knapsack problem $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$.
- $\varepsilon \geq 0$.

Process:

- Define
$$m \stackrel{\text{def}}{=} \left\lfloor \frac{n-1}{\varepsilon} \right\rfloor$$
- Compute S_0 as in the algorithm of 4.3.
- Build each S_{i+1} from S_i as follows:
  1. First compute S'_{i+1} as the full S_{i+1} of the standard dynamic programming algorithm.
  2. Define
$$\mathsf{Pr}_{i+1} \stackrel{\text{def}}{=} \max(\{p \mid \binom{p}{w} \in S'_{i+1}\})$$
  3. "Partition" S'_{i+1} into $m+1$ disjoint sets $\{C_1, C_2, \ldots, C_{m+1}\}$, with
$$C_k = \left\{ \binom{p}{w} \in S'_{i+1} \;\middle|\; \frac{k-1}{m} \cdot \mathsf{Pr}_{i+1} \leq p < \frac{k}{m} \cdot \mathsf{Pr}_{i+1} \right\}$$
     - Note that this is not a true partition, since some blocks may be empty.
  4. Create S_{i+1} from S'_{i+1} by retaining from each block C_k only the pair with the least weight.
- It is clear that this algorithm generates feasible solution, since pairs are only discarded; no new ones are generated.

10.5.3 Theorem

- *The algorithm of 10.5.2 is a fully polynomial approximation scheme for integer knapsack problems which runs in time $O(n^2/\varepsilon)$ for $\varepsilon \leq 1$.*

PROOF:

- Let the context be as in 10.5.2.
- The relative error in profit introduced at stage $i \geq 1$ is at most

$$\frac{\left(\frac{\mathsf{v}_i}{m}\right)}{\mathsf{Opt}(P)} \leq \frac{\left(\frac{\mathsf{v}_i}{m}\right)}{\mathsf{v}_i} = \frac{1}{m} = \frac{1}{\left\lfloor \frac{n-1}{\varepsilon} \right\rfloor} \leq \frac{\varepsilon}{n-1}$$

- The error at stage 0 is 0.
- Hence, the total error is bounded by

$$(n-1) \cdot \left(\frac{\varepsilon}{n-1} \right) = \varepsilon$$

  since the errors are additive in this algorithm.

- To establish the upper bound on the complexity, proceed as follows.
- At stage 0, there is one block.
- In every other stage, the maximum number of blocks is $m+1 = \lceil (n-1)/\varepsilon \rceil + 1$. Hence

$$\sum_{i=0}^{n-1} \mathsf{Card}(S_i) \leq 1 + (n+1) \cdot \left(\left\lfloor \frac{n-1}{\varepsilon} \right\rfloor + 1 \right) \in O(n^2/\varepsilon)$$

□

10.5.4 The idea behind separation

- In interval partitioning, the process may be thought of as one of first ordering the partial solutions at stage i by profit, and then drawing fixed boundaries to lump them together.
- These choices of boundaries may not be very good.
- In separation, a more dynamic approach to locating boundaries around clusters of values is employed.

10.5.5 The separation algorithm

Input:

- An n-element integer knapsack problem $P = (\mathsf{v}_{(-)}, \mathsf{w}_{(-)}, M)$.
- $\varepsilon \geq 0$.

Process:

- Compute S_0 as in the algorithm of 4.3.
- Build each S_{i+1} from S_i as follows:
  1. First compute S'_{i+1} as the full S_{i+1} of the standard dynamic programming algorithm.
  2. Define
  $$\mathsf{Pr}_{i+1} \stackrel{\text{def}}{=} \max(\{p \mid \binom{p}{w} \in S'_{i+1}\})$$
  3. Order the elements of S'_{i+1} by increasing profit.
  4. Scanning S'_{i+1} from left to right, build S_{i+1} is follows.
     (i) Retain the first element.
     (ii) Retain a subsequent element only if its profit exceeds that of the previously retained element by at least $\mathsf{Pr}_{i+1} \cdot (\varepsilon/(n+1))$.
- It is clear that this algorithm generates feasible solution, since pairs are only discarded; no new ones are generated.

10.5.6 Theorem

- *The algorithm of 10.5.5 is a fully polynomial approximation scheme for integer knapsack problems which runs in time $O(n^2/\varepsilon)$ for $\varepsilon \leq 1$.*

PROOF: The proof is similar to that of 10.5.3. □

10.6 Closing Comments

- The techniques presented here apply only to a small class of problems which are structurally similar to the knapsack problem.
- In general, approximation techniques are very problem specific.