Improved running time for DP.

Add an extra step 3 : dominated pairs are removed. A pair (t, w) dominates another pair (t', w') if $t \le t'$ and $w \ge w'$.

Theorem 3.1 The knapsack problem can be solved (exactly) in $O(n \cdot \min\{V, B\})$

Proof By removing each pair dominated by another pair, # of pairs in A_j is at most $\min\{B, V\}$. Thus step 1&2&3 can be done in $O(\min\{B, V\})$ time in each iteration.

Thus, the time is $O(n \cdot \min\{B, V\})$.

A PTAS for knapsack

1. Round the values $v_i' = \lfloor v_i / \mu \rfloor$, where $\mu = \frac{\varepsilon M}{n}$, $M = \max_i v_i$.
2. Apply the DP to the ~~second~~ rounded instance.

Analysis

$$v_i' = v_i/\mu = \frac{v_i \cdot n}{\varepsilon M} \le \frac{n}{\varepsilon}$$

- After rounding, we have $v_i' \in \{0, 1, \cdots, \lfloor n/\varepsilon \rfloor\}$
  - → DP runs in $O(nV') = O(n^3/\varepsilon)$ time where $V' = \sum_i v_i' = O(n^2/\varepsilon)$
  - → Polynomial time ↳ $\min(B, V') \le V'$.
- By rounding, the loss in precision is ① at most μ per item.
  - → loss is at most $n\mu = \varepsilon \max\{v_i\} = \varepsilon M \le \varepsilon OPT$.
  - → $(1-\varepsilon)$-approximation

Formal proof. Let S be the set of items found by DP and let O be an optimal set of items for the original (unrounded) instance.

For any item i, we have

① $v_i \ge \mu v_i' > \mu(v_i/\mu - 1) = v_i - \mu$

The value of the final solution is

$$\sum_{i \in S} v_i \overset{①}{\ge} \sum_{i \in S} \mu v_i' \overset{①}{\ge} \sum_{i \in O} \mu v_i' \overset{②}{\ge} \sum_{i \in O} (v_i - \mu)$$

$$\ge OPT - |O|\mu \ge OPT - n\mu = OPT - \varepsilon M$$

$$\overset{②}{\ge} OPT - \varepsilon OPT = (1-\varepsilon) OPT$$

① S is optimal for the rounded instance with value v_i'

② $OPT \ge M$ since taking the item with the largest value is a feasible solution.

3.2 Scheduling on identical machines.

Define: job j is long if $p_j > \varepsilon T^* \Rightarrow p_j > \varepsilon P/m$ (一台机器)

job j is short if $p_j \le \varepsilon T^* \Rightarrow p_j \le \varepsilon P/m$

choose T^* s.t. $T^* \le OPT$, for example $T^* = P/m$ with $P = \sum_j p_j$.

PTAS: step 1: Find a schedule for the long jobs of length at most $(1+\varepsilon)$ OPT ~~$C_{max} \le p_i + \sum_{j \ne i} p_j/m$~~

step 2: Add short jobs greedily.

Analysis: 1. Feasible ✓

2. Ratio $\le 1+\varepsilon$: ① the last job to complete is a short job.

$\sum_{j \ne i} p_j/m = S_i \le P/m \le OPT$. $C_{max} = S_L + p_L \le S_L + \varepsilon T^* \le OPT + \varepsilon OPT = (1+\varepsilon) OPT$.

② the last job to complete is a long job.

the makespan equals the length of the optimal schedule for just the long jobs, which is clearly no more than C_{max}^* for entire input.

3. Running time: **Lemma**: An optimal schedule of ~~the~~ long jobs can be found in $O(m^{m/\varepsilon})$ time.

Proof # of long jobs is at most $P/(\varepsilon T^*) = m/\varepsilon$.

To find the optimal one, try all possible schedules and take the best. Note the orders of jobs on a machine is not important for the length of the schedule.

Thus, # of possible schedules is at most $m^{m/\varepsilon}$.

Theorem 3.2 If m is assumed a constant, the PTAS runs in polynomial time.

The running time of the PTAS above is exponential in m. Next, we show how to get a PTAS with running time polynomial in m.

Ingredients:

- Partition in long and short
- Rounding of long jobs
- DP for long jobs
- Relaxed decision procedure

Define: job j is long if $p_j > \varepsilon T^*$
job j is short if $p_j \le \varepsilon T^*$, for some $T^* \le OPT$.

PTAS is same as before. Step 2 is easy, and need to find a different algo. for step 1. Choosing T^* is crucial.

Relaxed decision procedure

A simple tool for finding an α-approximation algo. for min problem.

Design an algo. with the following properties:

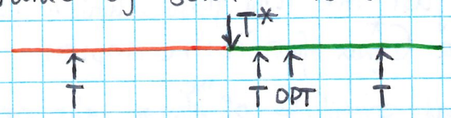
If $T \ge OPT$, it finds a solution of value at most αT

If $T < OPT$

Find the smallest T, say T^*, s.t. algo. finds a solution of value $\le \alpha T$

Then, we must have $T^* \le OPT$.

→ value of solution is at most $\alpha T^* \le \alpha OPT$



How to find T^*? (All numbers are integer).

1. Try all values: start with a large value and decrease by 1 each step.
(NOT Polynomial).

2. Binary search (polynomial)

idea: maintain lower bound LB, and upper bound UB, s.t. at any moment:

i) $OPT \ge LB$

ii) The solution of value $\le \alpha UB$

Binary search:

while $LB \ne UB$ do:

$T = \lfloor (LB+UB)/2 \rfloor$

If algo. gives solution of value $\le \alpha T$, then:

set $UB := T$

else: set $LB := T+1$

Return T

Running time: $\log(UB - LB)$, which is polynomial in the input size

Remember, we want a schedule for long jobs of length at most $(1+\varepsilon)OPT$.

We shall use the relaxed decision procedure and will.

design an algo. with following property:

If $T \ge OPT$, it finds a schedule for long jobs of length at most $(1+\varepsilon)T$.

We find the smallest T, say T^*, for which the algo. is successful.

Then, $T^* \le OPT$, and the length of schedule for long jobs is at most $(1+\varepsilon)T^* \le (1+\varepsilon)OPT$.

Improving Step 1.

Algo.(T): 1. Say job j is long of $p_j > \varepsilon T$.

2. Round down processing times of long jobs:

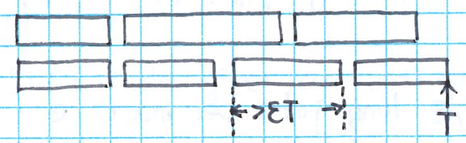
$p'_j = \lfloor p_j / \mu \rfloor \cdot \mu$, where $\mu = \varepsilon^2 / T$.

3. Find a schedule for rounded long jobs of length at most T (by DP)

4. Round up.

Lemma 1: If $T \ge OPT$, then the length of schedule is at most $(1+\varepsilon)T$.

Proof: Let σ be the schedule of length at most T computed in step 3.



1) There are at most $T/\varepsilon T = 1/\varepsilon$ long jobs per machine in σ.

2) Rounding up increase the length of each job by at most μ.

1)+2) → Rounding up increases the total length by at most $(1/\varepsilon)\mu = \varepsilon T$

Lemma 2: If $T \ge OPT$, then a schedule of length at most T can be found in polynomial time by DP.

Proof: D.P. (Note, if $T \ge OPT$, then such a schedule indeed exists since length are rounded down).

Some definition

- Note: Rounded processing times are a multiples of μ.

Say the job is of type t if its rounded processing time is $t\mu$

Let r be the # of types.

- Let n'_t be the number of long jobs of type t ($t=1,\cdots,r$)
- Let $(n_1, n_2, \cdots, n_r)$ represent a set of jobs with $n_t \le n'_t$ jobs of type t ($t=1,\cdots,r$)

  We call such a vector a configuration
- Let C be the set of all configuration that fit on a single machine i.e. the total job length at most T.

DP-table

For all configuration and # of machines $k \in \{1,\cdots,m\}$

Let $F_k(n_1,\cdots,n_r)$ = TRUE iff $(n_1,\cdots,n_r)$ can be scheduled on k machines in T.

Filling the DP-table

For all $(n_1,\cdots,n_r) \in C$, set $F_1(n_1,\cdots,n_r)$ = TRUE

For $k=2:m$:

  For all $(n_1,\cdots,n_r)$:

    if there is $(s_1,\cdots,s_r) \in C$, s.t. $F_{k-1}(n_1-s_1,\cdots,n_r-s_r)$=TRUE:

      Set $F_k(n_1,\cdots,n_r)$ = TRUE.

There is a feasible schedule iff $F_m(n'_1,\cdots,n'_r)$ = TRUE

Size of DP-table

From Lemma 1. at most $1/\varepsilon$ long jobs per machine

1) at most m/ε long jobs in total
2) at most m/ε of each type
3) at most $(m/\varepsilon)^r$ configurations $(n_1,\cdots,n_r)$ (with r the number of types)
4) at most $m(m/\varepsilon)^r$ entries $F_k(n_1,\cdots,n_r)$ in the DP-table

Rounded processing times are a multiple of μ.

5) at most $T/\mu = T/(\varepsilon^2 T) = 1/\varepsilon^2$ different processing times

   $r \le 1/\varepsilon^2$

4)+5) → DP-table is of polynomial size.

Total running time

- We showed that the number of entries in the DP-table is at most $m(m/\varepsilon)^r$ with $r \le 1/\varepsilon^2$.
- Computation of each entry takes $O(|C|)$ time, where, C is the set of all configuration $(n_1,\cdots,n_r)$ that fit on a single machine.
- $|C| \le (1/\varepsilon)^r$ since there are at most $1/\varepsilon$ long jobs per machine → $n_i \le 1/\varepsilon$

⇒ Total time is $O(m(m/\varepsilon)^r(1/\varepsilon)^r)$ with $r \le 1/\varepsilon^2$

which is polynomial in m, assuming that ε is constant.

Chapter 4. Deterministic Rounding of Linear Programs

4.1 Sum of completion times on a single machine.

Let OPT be the optimal value and let OPT^P be the optimal value for the preemptive problem. The optimal non-preemptive schedule is a feasible schedule for the preemptive problem.

$$OPT^P \le OPT.$$

The optimal preemptive schedule is founded by Shorest Remaining Processing Time (SRPT) rule.

Lemma The SRPT rule gives an optimal preemptive schedule.

Proof Assume a schedule σ does not satisfy the SRPT property, then there must be two jobs, say j and k, s.t. the pair j,k does not ~~sti~~ satisfy the SRPT property.

Let $t = \max\{r_j, r_k\}$. Assume that the remaining processing time of job j at time t is not larger than the remaining processing time of job k at time t.

Now schedule as follows: for every moment $s \ge t$ that σ processes j or k, now process job i at time s until j completes From that moment only process job k. Let C_j^σ and C_k^σ be the completion time in σ and let C'_j and C'_k be the new completion times. Then,