Chapter 3: Rounding Data and Dynamic Programming

(cp. Williamson & Shmoys, Chapter 3)

Knapsack Problem

Given: n items $I = \{1, \ldots, n\}$, values $v_i \in \mathbb{Z}_{>0}$ and sizes $s_i \in \mathbb{Z}_{>0}$, $i \in I$; knapsack of size $B \in \mathbb{Z}_{>0}$;

Task: find subset of items $S \subseteq I$ with $\sum_{i \in S} s_i \leq B$ maximizing $\sum_{i \in S} v_i$.

Remarks

- The Knapsack Problem is *NP*-hard (reduction from Partition).
- It can be solved in pseudo-polynomial time by dynamic programming (see below).

Notation:

- Subset of items $S \subseteq I$ has size and value $(t, w) := (\sum_{i \in S} s_i, \sum_{i \in S} v_i)$.
- Such a pair (t_1, w_1) is dominated by (t_2, w_2) if $t_1 \geq t_2$ and $w_1 \leq w_2$ and at most one of the two inequalities is tight;

Dynamic Program for Knapsack Problem

For $j = 0, 1, \ldots, n$ let $A(j)$ denote the set of feasible non-dominated pairs given by all subsets $S \subseteq \{1, \ldots, j\}$.

Dynamic Program

1. $A(0) := \{(0, 0)\}$;
2. for $j = 1, \ldots, n$ let $A(j) := A(j-1)$;
3. for each $(t, w) \in A(j-1)$
4. if $t + s_j \leq B$ then add $(t + s_j, w + v_j)$ to $A(j)$;
5. remove dominated pairs from $A(j)$;
6. return $\max_{(t,w)\in A(n)} w$;

Theorem 3.1.

The dynamic program correctly computes the optimal value of the knapsack problem in $O(n \cdot \min\{B, V\})$ time where $V := \sum_{i=1}^{n} v_i$. □

Remark: An optimal subset $S \subseteq I$ can be obtained by backtracking.

FPTAS for the Knapsack Problem

Definition 3.2 (FPTAS).

A fully polynomial-time approximation scheme is a PTAS $(A_\varepsilon)_{\varepsilon>0}$ such that the running time of A_ϵ is bounded by a polynomial in $1/\epsilon$.

FPTAS for Knapsack Problem

1. let $M := \max_{i\in I} v_i$; $\mu := \epsilon \cdot M/n$; $v'_i := \lfloor v_i/\mu \rfloor$ for $i \in I$;
2. solve knapsack instance with values v'_i by dynamic programming;

Theorem 3.3.

The algorithm above is a fully polynomial-time approximation scheme for the Knapsack Problem.

Proof:. . . □

Scheduling Jobs on Identical Parallel Machines

Given: n jobs $j = 1, \ldots, n$ with processing times $p_j \geq 0$, $j = 1, \ldots, n$, and m identical parallel machines.

Task: Process each job j nonpreemptively for p_j time units on one of the m machines. A machine can process at most one job at a time.

Remember:

- List scheduling in arbitrary order is a 2-approximation algorithm.
- List scheduling in LPT order is a 4/3-approximation algorithm.
- The analysis of both results relies on the fact that

$$C_{\max} \leq p_\ell + \frac{1}{m} \sum_{j \neq \ell} p_j \leq p_\ell + C^*_{\max}$$

  where ℓ is a job with maximal completion time $C_j = C_{\max}$.

PTAS for Constant Number of Machines

Let the number of machines m be constant and $\varepsilon > 0$ fixed.

Partition into short and long jobs:

- A job ℓ is called short if $p_\ell \leq \frac{\varepsilon}{m} \sum_j p_j$; otherwise, job ℓ is long.

Notice: There are at most $\lfloor m/\varepsilon \rfloor$ long jobs and this number is constant.

Algorithm A_ε

1. enumerate all schedules of long jobs; choose one with min makespan;
2. extend this schedule by using list scheduling for short jobs;

Theorem 3.4.

Algorithm A_ε runs in polynomial time and produces a schedule of makespan at most $(1 + \varepsilon) \cdot C^*_{\max}$.

Proof:. . . □

PTAS for Arbitrary Number of Machines

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m}\sum_j p_j$).

Short and long jobs:

- A job j is called short if $p_j \leq \varepsilon T$; otherwise, job j is long.
- For each long job j let $\bar{p}_j := \lfloor p_j/(\varepsilon^2 T) \rfloor \cdot \varepsilon^2 T$.
- Notice that there are at most $\lfloor 1/\varepsilon^2 \rfloor$ different rounded job sizes $\bar{p}_j$.

Algorithm B_ε

1. find schedule for all long jobs with rounded sizes $\bar{p}_j$ of makespan $\leq T$;
2. interpret as a schedule for the long jobs with original sizes p_j;
3. extend this schedule by using list scheduling for short jobs;

Analysis and Results

Theorem 3.5.

For a given problem instance and a target length T, Algorithm B_ε either correctly decides that there is no schedule of length $\leq T$ or it finds a schedule of length $\leq (1+\varepsilon) \cdot T$.

Proof:... □

Remarks:

- According to Theorem 3.5, Algorithm B_ε is a $(1+\varepsilon)$-approximate decision procedure.
- Together with a binary search framework for the optimal makespan $T = C^*_{\max}$, we get a polynomial-time approximation scheme (PTAS).

Theorem 3.6.

There is a polynomial-time approximation scheme for the problem of minimizing the makespan on a given number of identical parallel machines.

Proof:... □