CHAPTER 3 (3.1+3.2)

Rounding data and dynamic programming

Designing polynomial time approximation schemes

Introduction

A *polynomial-time approximation scheme* (PTAS) is a family of algorithms A_ϵ , where there is an algorithm for each $\epsilon > 0$, such that A_ϵ is a

- $(1+\epsilon)$-approximation algorithm (for minimization problems) or a
- $(1- \epsilon)$-approximation algorithm (for maximization problems).

A PTAS is the best you can hope for if the problem is NP-hard.

Introduction

- Running time depends on ϵ. (May be huge for small ϵ)
- Polynomial time only required for constant ϵ, e.g. $n^{(1/\epsilon)}$
- If also polynomial in $1/\epsilon$ then it is called an FPTAS, e.g. $(n/\epsilon)^2$

- Running time of PTAS usually very high.
- PTAS's are hardly ever applied in practice.
- PTAS gives theoretical evidence that approximation is easy.
- Problems which do have a PTAS are usually easy to handle in practice (by some other type of algorithms, e.g., local search)

- Designing a PTAS is usually done by standard techniques. However,
- algorithm and proof often contain many, many details.

Introduction

Two standard ingredients of a PTAS:

- Rounding data:
  - It simplifies the instance but reduces the precision. For example (divide by 10 and round down):

1023,	1026,	2061,	3,	12	→
102,	102,	206	0,	1	

- Dynamic Programming:
  - When instance is simplified enough, then straightforward DP may be enough to solve the simplified instance in polynomial time.

Knapsack (3.1)

Instance: Capacity B, and values v_i and size $s_i \leq B$ for each item $i \in I=\{1 \dots n\}$.

Solution: $S \subseteq I$ such that $\Sigma_{i \in S} s_i \leq B$

Value: $\Sigma_{i \in S} v_i$

Goal: Find a solution of maximum value.

- Knapsack is NP-hard but not *strongly* NP-hard since
- Knapsack can be solved in $O(n \cdot \min\{B,V\})$ time, where $V=\Sigma_i v_i$.

  (= pseudo polynomial time)

Knapsack

A pseudo polynomial algorithm by Dynamic Programming:

Define:

A_j : set of all pairs (t,w) such that there is a subset of items in {1,…, j} with size $t \le B$ and value w.

Then, OPT=maximum value of w in A_n

Algorithm:

$A_1=\{(0,0),(s_1,v_1)\}$ and for $j \ge 2$:

A_j : Step 1 $A_j \leftarrow A_{j-1}$

Step 2 For each $(t,w) \in A_{j-1}$: If $t+s_j \le B$ then add $(t+s_j, w+v_j)$ to A_j.

Knapsack

Running time of this DP

Each A_j has at most BV pairs (t,w) → Running time of DP is O(nBV).

Improved running time for DP

Say that pair (t,w) dominates another pair (t',w') if $t \leq t'$ and $w \geq w'$.

- Running time can be improved by removing dominated pairs.→

  Step 3: Remove dominated pairs from A_j.

Now, each A_j in DP has at most min{B,V} pairs: Tiime for DP is O(n min{B,V})

PTAS for Knapsack

PTAS

1. Round the values $v_i' = \lfloor v_i/\mu \rfloor$, where $\mu = \epsilon \cdot \max\{v_i\} / n$
2. Apply the DP (from previous slides) to the rounded instance

Analysis

- By rounding, $v_i' \in \{0,1,\ldots,n/\epsilon\}$
  - → DP runs in $O(nV') = O(n^3/\epsilon)$ time.
  - → = polynomial time.
- By rounding, the loss in precision is at most μ per item (see notes/book for proof)
  - → loss is at most $n\mu = \epsilon \max\{v_i\} \leq \epsilon$ Opt.
  - → $(1-\epsilon)$-approximation.

Scheduling on identical machines (3.2)

Load balancing problem

Instance:	n jobs with processing times $p_1, \ldots, p_n$ and m machines.
Solution:	Assignment of jobs to machines
Cost:	The maximum load over the machines (=length of schedule)
Goal:	Minimize length of schedule.

- Strongly NP-hard
- List scheduling gives (2-1/m)-approximation
- LPT gives 4/3-approximation

Scheduling on identical machines

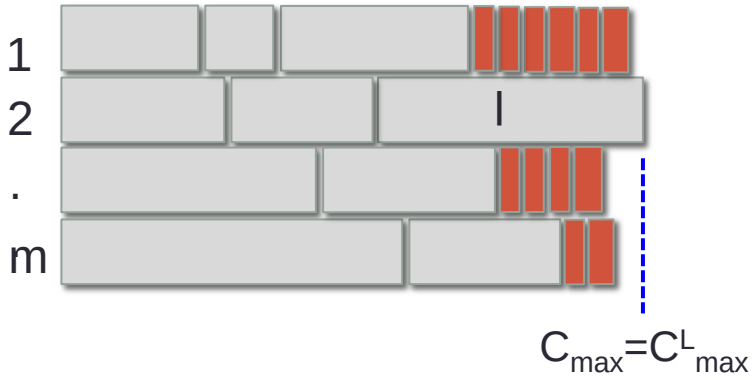
Idea for PTAS:

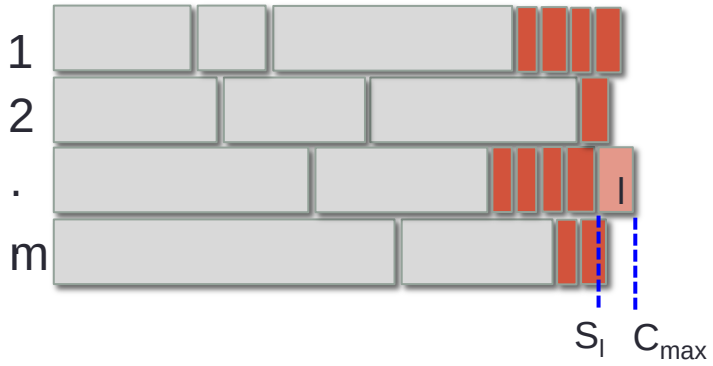
- Partition jobs in long and short
- Find an (almost) optimal schedule for long jobs
  - Idea: Running time may be OK if only few long jobs.
- Add short jobs greedily
  - Idea: Since jobs are short, this will only give a small error.

Scheduling on identical machines

Idea for PTAS:

- Partition jobs in long and short
- Find an (almost) optimal schedule for long jobs
  - Idea: Running time may be OK if only few long jobs.
- Add short jobs greedily
  - Idea: Since jobs are short, this will not give a large error.





Scheduling on identical machines

- `Long' should be long enough to have few of them
- `Short' should be short enough to keep error small.

Choose T^* such that $T^* \leq OPT$. Say

- job j is long if $p_j > \epsilon T^*$
- job j is short if $p_j \leq \epsilon T^*$ $(\leq \epsilon OPT)$

PTAS

Step 1 Find a schedule for the long jobs of length at most $(1 + \varepsilon)OPT$

Step 2 Add short jobs greedily

Analysis

i. Feasible schedule? ✓

ii. $(1+\epsilon)$-approximation? ✓

iii. Running time? Step 2 ✓ , Step 1 ??

Scheduling on identical machines

Running time of Step 1?

How about complete enumeration?

- Denote $P=\Sigma_j p_j$ and take for example $T^*=P/m$ (Then, $T^* \leq OPT$)
- At most $P/(\epsilon T^*) = m/\epsilon$ long jobs
- Each long job needs to be assigned to one of the m machines
- → at most $m^{(m/\epsilon)}$ possible schedules. (exponential in m)

Theorem

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

☹ $m^{(m/\epsilon)}$ time is pretty slow! Take for example $m=10$, $\epsilon=0.1$ → 10^{100}

Scheduling on identical machines

- We have seen:

  A PTAS for scheduling on identical machines assuming m is a constant.

  However, this PTAS is slow even for relatively small m.

- We want:

  A PTAS which is polynomial in n and m .

  Next slides …

Scheduling on identical machines

Keep same outline:

Choose T^* such that $T^* \leq OPT$. Say

- job j is long if $p_j > \epsilon T^*$
- job j is short if $p_j \leq \epsilon T^*$ $(\leq \epsilon OPT)$

PTAS

Step 1 Find a schedule for the long jobs of length at most $(1 + \varepsilon)OPT$

Step 2 Add short jobs greedily

- Step 2 is easy.
- Need to find a different algorithm for Step 1.

General technique: Relaxed decision procedure

‘Simple’ way to find an α–approximation algorithm for a minimization problem:

General technique: Relaxed decision procedure

'Simple' way to find an α–approximation algorithm for a minimization problem:

Design an algorithm which finds a solution of value ≤ αT, if OPT≤ T.

- The algorithm can do anything if OPT > T (we don't care).

Now,

- Find smallest T, say T*, such that the algorithm gives a solution of value ≤ αT.
- → then T* ≤ OPT.
- → value ≤ αT*≤ αOPT.

General technique: Relaxed decision procedure

How to find smallest T? (assume integer values)

1. **Try all values (not polynomial in general)**

2. **Binary search (= polynomial)**

Maintain lower bound LB and upper bound UB such that at any moment
- OPT ≥ LB and
- we have a solution of value ≤ αUB.

Binary search:
While LB ≠ UB do
T = ⌊(UB+LB)/2⌋
If algorithm gives solution of value ≤ αT then set UB:=T,
otherwise, set LB:=T+1
Output T*=T

Running time: $^{2}\log(UB-LB)$ which is polynomial (for almost any problem).

Scheduling on identical machines

Design algorithm which, for any T, finds a schedule of the long jobs of length

$\leq (1+\epsilon)T$, if $OPT \leq T$.

Then:

Find the smallest T such that the algorithm finds a schedule of length $\leq (1+\epsilon)T$.
Let T* be this smallest value.

Result:

- $T^* \leq OPT$ and
- We have a schedule for long jobs of length $\leq (1+\epsilon)T^* \leq (1+\epsilon)OPT$

Now add small jobs in a greedy way. → ✓ PTAS

Scheduling on identical machines

Design algorithm which, for any T, finds a schedule of the long jobs of length

$\leq (1+\epsilon)T$, if $OPT \leq T$.

How? Rounding + DP:

Round processing times:

$\mu = \epsilon^2 T$ and $p_j' = \lfloor p_i/\mu \rfloor \mu$

Algorithm:

1. Find optimal schedule of rounded long jobs (by DP)
2. Use this as solution for the unrounded long jobs.

if $OPT \leq T$

Lemma Length of schedule is at most $(1+\epsilon)T$

Proof

- See notes.

Scheduling on identical machines

Optimal schedule of rounded long jobs can be found in polynomial time by DP:

Outline

i. Number of long jobs per machine is at most $1/\epsilon$ (=const.)
ii. Number of different processing times, say r, is at most $1/\epsilon^2$ (=const.)
iii. i+ii → no more than $(1/\epsilon^2)^{(1/\epsilon)}$ configuration per machines (=const.).

DP will work if we fill the DP table machine by machine: Start with 1 machine and end with m machines.

Details next slide …

Scheduling on identical machines

Notation:

- $(n_1,n_2,\ldots,n_r)$ represents a set of jobs with n_i jobs of type i for i=1…r
- set C contains all sets $(n_1,n_2,\ldots,n_r)$ for which the total processing time is at most T
- Let $(n'_1,n'_2,\ldots,n'_r)$ be the number of long jobs of each type.

D.P.

$F_k(n_1,n_2,\ldots,n_r)$ = TRUE if $(n_1,n_2,\ldots,n_r)$ can be scheduled on k machines within time T.

Schedule found if $F_m(n'_1,n'_2,\ldots,n'_r)$ =TRUE

k=1: $F_1(n_1,n_2,\ldots,n_r)$ = TRUE if $(n_1,n_2,\ldots,n_r) \in C$

For $2 \leq k \leq m$

$F_k(n_1,n_2,\ldots,n_r)$ = TRUE if there is a $(s_1,\ldots,s_r) \in C$ s.t. $F_{k-1}(n_1-s_1, \ldots, n_r-s_r)$ =TRUE

Scheduling on identical machines

Running time

At most $1/\epsilon$ long jobs per machines

→ at most m/ϵ long jobs in total

→ at most m/ϵ of each type

→ at most $(m/\epsilon)^r$ vectors $(n_1,n_2,\ldots,n_r)$

→ at most $m(m/\epsilon)^r$ entries $F_m(n_1,n_2,\ldots,n_r)$ in DP table.

Evaluation of each entry takes $|C|$ time

→Total time: $|C|m(m/\epsilon)^r$, where $|C| \leq (1/\epsilon^2)^{(1/\epsilon)}$ and $r \leq 1/\epsilon^2$.

→= polynomial in m (but not polyn. in $1/\epsilon$). So, we have our **PTAS** (but not an FPTAS).