

## Lecture 3 — 1/15/2013

*Prof. Russell Impagliazzo**Scribe: Christopher Tosh*

## Overview

We are covering various techniques for analyzing approximation algorithms. There are four basic techniques:

- *Reduction Method*
- *Sacrifice Precision Method*
- *Combine Bounds Method*
- *Transformation Method*

In the previous two classes we covered the reduction and sacrifice precision methods. In this class, we will consider an application of the sacrifice precision method and discuss the combine bounds method.

## Sacrifice Precision Method

Recall that the sacrifice precision method is used in the context where the difficulty of the problem arises from the degree of precision required to compute exactly the optimal solution. The solution is to solve a simpler instance of the problem exactly and hope that the solution is not too far from the optimal solution of the original instance.

### Load Balancing on $m$ identical machines

The setup is as follows

**Instance:** We are given  $n$  jobs, each with weight  $d_1, d_2, \dots, d_n$  and  $m$  machines

**Solution:** We must assign each job  $i$  to a machine  $j$

**Objective:** Minimize the maximum load on any of the  $m$  machines

We saw last class that if there are only  $k$  types of jobs, we can use dynamic programming to get a solution in  $O(n^{2k}m)$  time. We will transform the original problem into this problem using the sacrifice precision method to obtain a PTAS.

Let  $1 + \epsilon$  be the approximation ratio we desire, and let  $\delta = \epsilon/2$ . Let  $D$  be the longest duration of any of our jobs, that is  $D = \max_i d_i$ . Then let

$$\begin{aligned} D_0 &= D \\ D_1 &= D/(1 + \delta) \\ D_2 &= D/(1 + \delta)^2 \\ &\vdots \\ D_k &= D/(1 + \delta)^k \end{aligned}$$

Where if  $k = \frac{1}{\delta} \ln \frac{1}{\delta}$ , then  $D_k \approx \delta D$ . Then our algorithm proceeds as follows:

1. Temporarily delete all jobs  $j$  with  $d_j < D_k$ .
2. Round up each remaining  $d_j$  to the smallest  $D_i$  s.t.  $D_i \geq d_j$ . Call this  $\hat{d}_j$ .
3. Solve the  $\hat{d}_j$  problem optimally to get an assignment for the large jobs.
4. For each small  $d_j$ , assign  $j$  to the currently least loaded machine.

Note that the way the  $D'_i$ s are defined means that for each  $\hat{d}_j$ , we have  $d_j \leq \hat{d}_j \leq (1 + \delta)d_j$ .

Let  $I_1$  denote the original problem instance, i.e. the problem corresponding to the jobs with weights  $d_1, \dots, d_n$ . Let  $OPT_1$  denote the solution of the optimal cost,  $S_1$  correspond to our solution, and  $A_1$  correspond to the the cost of  $S_1$  for the problem instance  $I_1$ .

Let  $I_2$  denote the problem instance for all jobs whose cost is at least  $D_k$ , i.e. the subset of jobs  $j$  with weights  $d_j \geq D_k$ . Let  $OPT_2$ ,  $S_2$ , and  $A_2$  be defined correspondingly for  $I_2$ .

Let  $I_3$  denote the transformed problem instance of  $I_2$ , i.e. the set of jobs with weights  $\hat{d}_j$ . Let  $OPT_3$ ,  $S_3$ , and  $A_3$  be defined correspondingly for  $I_3$ .

Note that since we solve  $I_3$  optimally,  $A_3 = OPT_3$ . Also,  $S_3 = S_2$ , since we simply use the solution of  $I_3$  for  $I_2$ . Finally,  $S_1 = S_2 + \text{something for the small jobs}$ .

**Lemma 1.**  $(1 + \delta)OPT_2 \geq OPT_3 \geq OPT_2$

*Proof.* Since any machine in  $OPT_2$  can have their cost multiplied by at most  $1 + \delta$ , the first inequality holds. The second inequality holds since each machine in  $I_2$  has their cost rounded up in  $I_3$ .  $\square$

Now we can use this lemma to talk about the solution cost of  $S_2$ .

**Lemma 2.**  $A_2 \leq (1 + \delta)OPT_2$

*Proof.* Since the weights of the solution  $S_2$  are at most the weights of the solution  $S_3$ , we have  $A_2 \leq A_3$ . However, we know that  $A_3 = OPT_3 \leq (1 + \delta)OPT_2$ .  $\square$

Now, let  $i$  be the most loaded machine in  $S_2$ . Then there are 2 cases.

**Case 1:** If in  $S_1$  we never schedule a small job on machine  $i$ , then

$$\begin{aligned} A_1 &\leq A_2 + D_k \\ &\leq (1 + \delta)OPT_2 + D_k \\ &\leq (1 + \delta)OPT_1 + \delta D \\ &\leq (1 + \delta)OPT_1 + \delta OPT_1 \\ &\leq (1 + 2\delta)OPT_1 \end{aligned}$$

**Case 2:** If we do schedule a small job on  $i$  in  $S_1$ , then we've scheduled small jobs on every machine. Thus the last job on the heaviest loaded machine was a small job, implying that every machine has load  $\geq A_1 - D_k$ .

Thus, the total sum of work across machines  $T$  is at least  $m(A_1 - D_k)$ . Then we have

$$A_1 \leq \frac{T}{m} + D_k \leq OPT_1 + \delta D = (1 + \delta)OPT_1$$

where the second inequality follows from the fact that the even distribution of work across all machines is a lower bound on the optimal solution.

Thus, our algorithm produces a  $(1 + 2\delta) = (1 + \epsilon)$ -approximation. The running time of this algorithm is  $O(n^{2k}m) = O(n^{\frac{2}{\delta} \log \frac{2}{\delta} m})$ .

## Combine Bounds Method

The combine bounds method arises when we can establish a set of bounds for some optimization problem and then demonstrate that our algorithm's solution falls within some linear combination of these bounds. To demonstrate this, we will consider the two following problems.

### Parallel Scheduling

The setup is as follows

**Instance:** We have  $P$  processors and  $n$  jobs, each with duration  $d_j$  and processor requirement  $1 \leq p_i \leq P$ .

**Solution:** We must assign each job  $j$  a time interval  $(s_j, f_j)$  of length  $d_j$ .

**Constraint:** At any time  $t$ , the total processor requirement cannot be more than  $P$ .

**Objective:** Minimize the final completion time of any job.

Here is a simple greedy algorithm:

1. Sort jobs by processor requirement in decreasing order.
2. Start each job after the earlier ones have been scheduled and we have enough processors.

Let  $t_0$  denote the first time that we have more than  $P/2$  idle processors and  $t_1$  be the time we finish.

**Lemma 3.** *At  $t_0$ , all jobs have been started.*

*Proof.* While there are jobs  $i$  with  $p_i \geq P/2$ , we run such jobs. Thus at time  $t_0$ , all jobs satisfy  $p_j < P/2$ . However, if at time  $t_0$  there are more than  $P/2$  idle processors and we have not started a job, we would schedule that job. Thus, at  $t_0$  all jobs have been started.  $\square$

Thus,  $t_1 \leq t_0 + \text{duration of the last job to finish}$ .

Let  $W = \sum_{i=1}^n p_i d_i$  denote the total work that is to be done. Then the following lower bounds hold trivially.

**Lemma 4.**  $OPT \geq W/P$

**Lemma 5.**  $OPT \geq \max_i d_i$

The total work done by our algorithm at time  $t_0$  is at most all the total work. However, since up until  $t_0$  we have always kept at least  $P/2$  processors working, we have

$$W \geq \text{Work done by time } t_0 \geq \frac{P}{2} t_0$$

Thus, we obtain  $t_0 \leq 2\frac{W}{P} \leq 2OPT$ . Therefore, our algorithm's solution is upper-bounded by

$$\begin{aligned} t_1 &\leq t_0 + \text{duration of last job} \\ &\leq t_0 + \max_i d_i \\ &\leq 2OPT + OPT \\ &= 3OPT \end{aligned}$$

## Set Cover Problem

The setup is as follows

**Instance:** We have a universe  $U = \{u_1, \dots, u_n\}$  and a family of subsets  $S_1, \dots, S_m \subset U$  s.t.  $|S_i| \leq k$

**Solution:** We choose subsets  $S_{j_1}, \dots, S_{j_l}$  s.t.  $\bigcup_{i=1}^l S_{j_i} = U$

**Objective:** Minimize the number of subsets we choose.

A trivial solution is to iteratively pick a new set that is not in the union of the sets we have chosen, i.e. to always cover at least one new element when we choose a set. Then we are guaranteed to pick at most  $n$  sets. Since the best we can do is cover  $k$  elements with every set we choose, this is a  $k$ -approximation.

We want to get an approximation ratio of  $H_k = \sum_{i=1}^k 1/i \approx \ln(k + c)$  for some constant  $c$ .

Suppose that we assign weights  $w_i$  to each element  $i$ . Then for any assignment of weights,

$$l \geq \frac{\sum w_i}{\max_i \sum_{x \in S_i} w(x)}$$

Our algorithm will simultaneously assign weights and choose sets. It proceeds as follows:

1. Find the set  $S_i$  that covers the most new elements (say  $r$  new elements).
2. Assign the newly covered elements weight  $1/r$ .
3. Repeat from step 1 until all elements are covered.

Let  $l$  be the number of sets in our set cover and  $W = \sum_x w(x)$ . Then  $W = l$ . Thus, our task is reduced to showing that  $\max_i \sum_{x \in S_i} w(x) \leq H_k$ .