Complexity and Advanced Algorithms

Spring 2021

Approximation Algorithms – A Brief Introduction

Approximation Algorithms

- Suppose a problem is known to be NP-Complete.
- No hope to solve in polynomial time unless P = NP.
- But, several practical problems fall in this category.
- Need some solution to this issue.
- This is where approximation algorithms help.

Approximation Algorithms

- For a problem P, let A be an approximation algorithm.
- Suppose that the problem P is a minimization problem.
- Then, the performance of Algorithm A for P is measured as its approximation ratio defined as follows.

value of the

- For an instance I of P, let OPT(I) denote the best possible solution.

the value of optimal

- Let A(I) denote the solution produced by the algorithm A.
- The approximation ratio of algorithm A is $\max_I |A(I)|/|OPT(I)|$.
- Notice that the ratio is always at least 1.

TSP(G) {
:
return Tour
}

OPT(I) ≤ A(I)

Length(Tour)

Alg

Approximation Algorithms

- For a problem P, let A be an approximation algorithm.
- Suppose that the problem P is a maximization problem.
- Then, the performance of Algorithm A for P is measured as its approximation ratio defined as follows.
- For an instance I of P, let OPT(I) denote the best possible solution.
- Let A(I) denote the solution produced by the algorithm A.
- The approximation ratio of algorithm A is $\boxed{\max_I |OPT(I)|/|A(I)|}$.
- Notice that the ratio is always at least 1.

OPT(I) ≥ A(I)

Approximation Algorithms

- We have seen an example of this definition earlier.
- In the context of MAXSAT.
- Today, we will study two more problems and approximation algorithms for them.

MAX $f(\ )$

s.t. ,

Best possible I : $f(OPT(I)) = 20$

for Any alg. $f(A(I)) \leq 20$

for any input, if $\frac{|OPT(I)|}{|A(I)|} \leq \frac{3}{2}$

$|A(I)| \geq \frac{2}{3} |OPT(I)|$

3/4 for MAXSAT

Load Balancing

- Consider m machines $M_1, M_2, \ldots, M_m$, that are identical in all respects.
  - Like the m cores of your multicore computer.
- We have n jobs, $J_1, J_2, \ldots, J_n$ to be processed and any job can be processed by any machine.
- Our goal is to minimize the time spent by any machine.

6 Jobs

4
2
5
3
7
2

4 cores

M_1	M_2	M_3	M_4
J_1	J_3	J_5	J_2
J_4	J_6		
7	7	7	2

min

Load Balancing

- We define the following quantities.
- Let A(i) be the jobs assigned to machine M_i.
- Job j has a time requirement t_j, for j = 1 to n.
- The makespan of machine M_i is $T_i := \Sigma_{j \in A(i)} t_j$.

- The makespan of an assignment $T = \max_i T_i$.
- The goal of the problem is to find an assignment that minimizes the makespan.

Load Balancing

- One of the popular algorithms is to use a greedy technique.
- We can assume that all the jobs are given apriori.
  - A bit unlike the real world setting where user jobs are fired any time.
- Assign the next job to the machine that is presently least loaded.
  - Note that all jobs are given at the beginning.
  - Assignment is done before any machine starts its processing.

Load Balancing

- GreedyAssign(Jobs J, m)
- begin
  - Set A(i) = empty for all i between 1 to m.
  - Set $T_i = 0$ for all i between 1 to m
  - For j = 1 to n do
    - Find an index i such that machine M_i has minimum T_i
    - A(i) = A(i) U {j}
    - $T_i = T_i + t_j$ load
  - Endfor
- End

J_1 J_2 J_3 . . . J_n

4 2 3 5

M_1 M_2 .. M_m

load

j: Jobs

i: machines

9

M_1	M_2
4	2
5	3
9	5

Load Balancing

- Illustrate how this algorithm works on the following set of jobs and four machines.

- Runtimes of jobs = 2, 3, 2, 2, 4, 1, 2, 1.

Some machine

makespan ≥ 4

max makespan ≥ 4

- Find the best possible makespan and the makespan produced by the greedy algorithm.
  - For j = 1 to n do
    - Find an index i such that machine M_i has minimum T_i
    - A(i) = A(i) U {j}
    - $T_i = T_i + t_j$
  - Endfor

greedy : 6

best : 5

Load Balancing

- We start with two observations that help us prove the approximation ratio of the greedy algorithm.
- Let T^* denote the best possible makespan. (Ex: $T^* = 5$)

$T^* \geq \frac{15}{4}$

- Observation 1: $T^* \geq (1/m) \Sigma_{j=1}^{n} t_j$.

$= 15$ (ex.)

- Comes from the fact that there is a total work of $\Sigma_{j=1}^{m} t_j$, and some machine will have to work for at least a 1/m fraction of the total.

$J_1 \quad J_2 \quad J_3 \quad J_4 \quad J_5 \qquad m = 3$

$1 \quad 1 \quad 100 \quad 1 \quad 1$

- Observation 2: $T^* \geq \max_j t_j$.

Obs 1 $\Rightarrow$ $T^* \geq \frac{100+4}{3}$

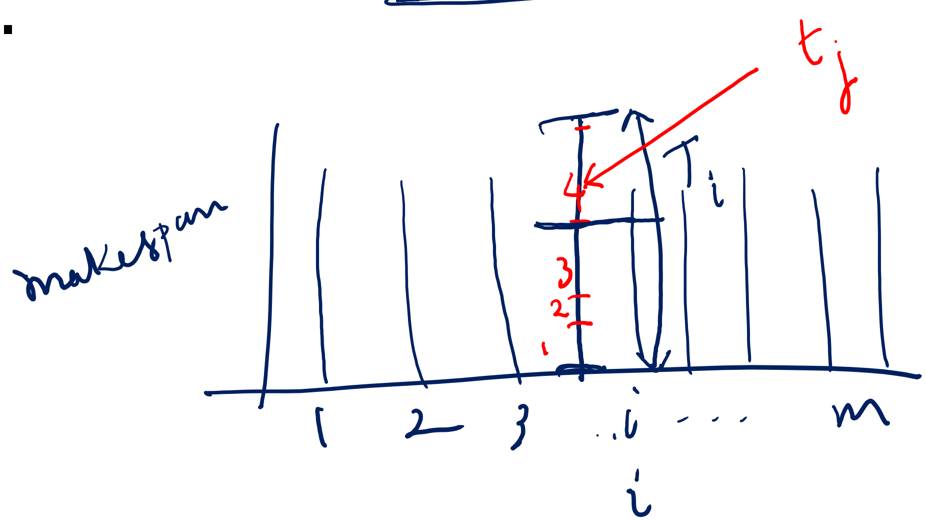
  - The longest job will be on some machine which will work for at least that much time.

≈ 34.6

Obs 2: $T^* \geq 100$

Load Balancing

- Let us now turn our attention to the greedy algorithm.
- Consider the machine that has the largest makespan according to the assignment produced by the greedy algorithm.
- Let this be the machine M_i, and its makespan be T_i.
- Let t_j be the last job assigned to M_i.
- Why did we assign t_j to M_i?
  - As M_i has the least load just before assigning t_j to M_i.



$A(I) = 6$

$OPT(I) = 5$

Load Balancing

$\frac{|A(I)|}{|OPT(I)|}$ ← ub

- The load of M_i before assigning T_j is $T_i - t_j$.
- At this time, every other machine has a load at least $T_i - t_j$. lb
  - Plus any other jobs assigned to them later on.
- So, the sum of the time for all jobs is lower bounded as follows.

sum of durations of jobs up to job j

- $\Sigma_{j=1}^{n} t_j \geq m(T_i - t_j)$, or

sum of job durations

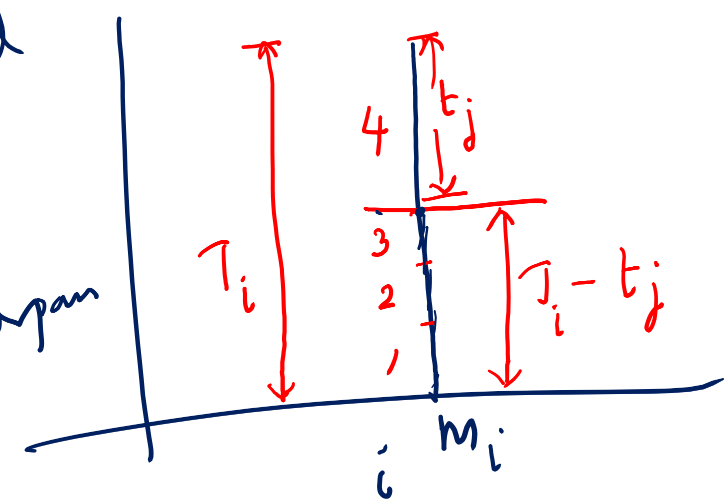
global

- $T_i - t_j \leq \underbrace{(1/m)\, \Sigma_{j=1}^{n} t_j}_{} \leq T^*$.

makespan

M_i rearrange as

by obs (1)



Load Balancing

- Further, notice that $t_j \leq T^*$.
- Use Observation 2.

$t_j \leq \max_k t_k \leq T^*$

Obs(2)

- Therefore, $T_i = T_i - t_j + t_j = \underbrace{(T_i - t_j)}_{} + (t_j) \leq T^* + T^* = 2T^*$.
- Hence, $T_i \leq 2T^*$.

$$\frac{|A(I)|}{|OPT(I)|} = \frac{T_i}{T^*} \leq \frac{2T^*}{T^*}$$

$$\leq 2$$

Load Balancing

- What would be one way to improve the solution produced by the greedy algorithm?
- Sort the jobs in descending order according to their runtime and assign them just as earlier.

- Let us call this as SortedGreedyAssignment.
- We will now analyze this approach.

Load Balancing

- Convince yourself that also SortedGreedyAssignment does not produce the best possible makespan.

Load Balancing

- The proof is not much different from the earlier proof.
- We will obtain a better bound for t_j, the last job assigned to the machine M_i that eventually has the largest makespan, T_i.
- Consider the case where there are fewer than m jobs.

$n \le m$

- Then, the best possible assignment is to give each job to a different machine.

$m = 4$

$n = 4$

  - Our algorithm also does the same.
  - So, we indeed produce the best possible makespan.

$\left\{ \begin{matrix} J_1 & J_2 & J_3 & J_4 \\ M_1 & M_2 & M_3 & M_4 \end{matrix} \right\}$

Load Balancing

- Let us consider the case that n is more than m.
- More jobs than machines.

$n > m$

decreasing order of t_j

- In any assignment, the job t_{m+1} in sorted order, is given to some machine that already has one additional job that is at least as long as t_{m+1}.

$t_1 \geq t_2 \geq t_3 \geq \cdots \geq t_m \geq t_{m+1} \geq \cdots \geq t_n$

- So, $T^* \geq 2t_{m+1}$.

best possible makespan

$t_{m+1} \leq t_i$

J_{m+1}

$J_1 \quad J_2 \quad \cdots \quad J_i \quad J_m$

$M_1 \quad M_2 \quad \cdots \quad M_i \quad \cdots \quad M_m$

Load Balancing

- Further, note that for machine M_i that has the largest makespan in our algorithm, if t_j is the last job assigned to M_i, then $t_j \le t_{m+1}$.
- From the previous, $t_{m+1} \le 1/2\ T^*$.
- Now, $T_i = T_i - t_j + t_j \le T^* + 1/2\ T^* = 3/2\ T^*$.

$t_j \le t_{m+1}$

as $j \ge m+1$ &

duration in sorted order

by our algorithm

obser (1)