CSL 3010: Design and Analysis of Algorithms July-November 2022

Lecture 10: Approximation Algorithms I

Lecturer: Dr. Saket Saurabh *Scribe: Mukul, Sawan, Jaimin, Tanay*

1 Introduction to Approximation Algorithms

We have seen till now that some algorithms would never be solved in polynomial time, but in some cases, they are critical to many real-life applications and implementations. So we came up with some algorithms which though do not give optimal solutions to the problem, give quite convincing and close (approximate) solutions.

Definition :An approximation technique gives a result that is demonstrably close to the ideal (as opposed to a heuristic that may or may not find a good solution). Approximation algorithms can also be utilized in some cases when a rapid near-optimal solution can be obtained and a precise answer is not required. Approximation techniques are often employed when finding an ideal solution is difficult.
Sometimes there is confusion between heuristics algorithms and approximated algorithms.

Heuristics algorithms	Approximated algorithms
Develop intuitive algorithms.	Guaranteed to run in polynomial time.
Guaranteed to run in polynomial time.	Guaranteed to get a solution which is close to the optimal solution
No guarantees on the quality of the solution.	Need to prove a solution's value is close to an optimum value, without even knowing what the optimum value is!

2 Minimum Makespan

Input : Set and n jobs and m machines, Array T (T[j] = Running time of job j)
Output : Job Assignment to machines.
Objective Function :

$$\text{MinimumMakespan(A)} = \max \sum_{A[j]=i}^{n} T[j]$$

Consider the following be the makespans of the jobs :
$[Job_1, Job_2, Job_3, Job_4, Job_5, Job_6, Job_7] = [2, 5, 7, 3, 6, 8, 4]$

If we consider the following allocation :
$\text{Job}_1, Job_2, Job_3 \rightarrow Machine_1$
$Job_4, Job_5 \rightarrow Machine_2$
$Job_6, Job_7 \rightarrow Machine_3$

Applying the minimumMakespan formula to the above allocation, we get:
$\text{MinimumMakespan(A)} = \max(\text{Job}_1 + Job_2 + Job_3, Job_4 + Job_5, Job_6 + Job_7) = max(14, 9, 12) = 14.$

Our job is to minimize this value.

2.1 Greedy Load Balancing Method

Approach : We will approximate this via the Greedy approach. Consider each job one at a time and assign each job to a machine i with the earliest finishing time. The consideration of the earliest finishing time makes our approach greedy.
We consider appending the task for a machine that completed its task to maximize the number of jobs to b done. This is also called greedy load balancing.

Pseudo code:

```
def GREEDY_LOAD_BALANCING(T, m, n):

    for i in range(1, m+1):
        sol[i] = 0

    for j in range(1, n+1):
        minFinish = min(sol)
        A[j].starting_time = minFinish
        sol[minFinish] += T[j]

    return A
```

Listing 1: Pseudo Python code for Greedy Load Balancing for Minimum Makespan

When we apply this approach to solve the minimum makespan problem, we get the following allocation:

$\text{Job}_1, Job_4, Job_5 \rightarrow Machine_1$
$Job_2, Job_6 \rightarrow Machine_2$
$Job_3, Job_7 \rightarrow Machine_3$

Applying the minimumMakespan formula to the above allocation, we get:
$\text{MinimumMakespan(A)} = \max(\text{Job}_1 + Job_4 + Job_5, Job_2 + Job_6, Job_3 + Job_7) = max(11, 13, 11) = 13$

This clearly shows that we have made a significant improvement in approximating the minimum Makespan problem. But to what extent? To know that we will have to analyze and try to quantify that. Now, we'll try to quantify that how much our approximation is far from the optimal solution.

2.2 Estimation Factor of Greedy Load Balancing Algorithm

Definition : **Approximation Factor** : If we can show that for any instance of the problem returned by our algorithm divided by the optimal solution of the instance is less than α, this α is called approximating factor.

$$\forall(ins), \frac{Algorithm(ins)}{OPT(ins)} \leq \alpha$$

Here, OPT(ins) denotes the algorithm of optimal assignment of the ins.

Now, we'll prove that the Minimum Makespan has an approximating factor of 2. For that, we can derive an expression similar to the above where α is 2.

Lemma 1:

$$\text{OPT(ins)} \geqslant max_j(T[j])$$

Proof: The optimum solution can never be less than the maximum of the spans of jobs.

Lemma 2:

$$\text{OPT(ins)} \geqslant \frac{1}{m} \sum_{i=1}^{n} T[i]$$

Proof: The optimum solution can never be less than the average of the spans of jobs.

Lemma 3:

$$\text{minimumMakespan(A)} \leq max_j(T[j]) + \frac{1}{m} \sum_{i=1}^{n} T[i]$$

Proof: Take machine j and its highest load, T[j]. Let task i be the final one on machine j. j may have had, a lighter load than the norm because he had the lowest load when I was scheduled. We've got Total[i] = (Total[i] - T[i]) + T[i] $\leq max_j(T[j]) + \frac{1}{m} \sum_{i=1}^{n} T[i]$

Combining all the three lemmas above will imply the greedy load balancing.

3 Room for Improvement

We are finding minimum everytime we run the program of greedy balancing. Here, we can sort T in decreasing order and get some improvement over this.

```
def Sorted_Load_Balancing(T, m, n):

    reverse_sort(T)
    min_makespan = GREEDY_LOAD_BALANCING(T, m, n)
    return min_makespan
```

Listing 2: Pseudo Python code for Sorted Greedy Load Balancing for Minimum Makespan

3.1 Calculation of Approximation Factor

If there are less than or equal to m jobs, then the above method is optimal.
But if the number of jobs is more than m, then at least one machine would get 2 of the first m+1 jobs. We can see that all the jobs are more than or equal to T[m+1].

$$\text{OPT(ins)} \geqslant 2T[m+1]$$

Let i be the last job assigned to the allocation.As we have sorted the makespans of the jobs, therefore,

$$\text{T[i]} \leqslant T[m+1] \leqslant \frac{OPT(ins)}{2}$$

So we would get,

$$\begin{aligned} \text{T} &= (\text{T - T[i]}) + \text{T[i]} \\ &\leqslant \frac{1}{m} \sum_{i=1}^{n} T[i] + T[i] \\ &\leqslant OPT(ins) + \frac{OPT(ins)}{2} \\ &\leqslant \frac{3}{2} OPT(ins) \end{aligned}$$

Here, we showed that the approximation factor of this sorted greedy load balancing method is 3/2.

3.2 A further Improvement

Proof of this improvement is out of the scope of the lecture.
If we are able to prove that The most recent task given to the machine carrying the most load has an index lower than 2m, then we would get,

$$\begin{aligned} \text{T} &= (\text{T - T[i]}) + \text{T[i]} \\ \text{T} &\leqslant \frac{1}{m}\sum_{i=1}^{n} Total[i] + T[2m+1] \\ &\leqslant OPT(ins) + \frac{OPT(ins)}{3} \\ &\leqslant \frac{4}{3}OPT(ins) \end{aligned}$$

Here, we showed that the approximation factor of this sorted greedy load balancing method is 3/2.