Fall 2016

Project Report

Mahesh Kumar Gupta
Ajay Vijayakumar
Maulik Rawal

Weighted Interval Scheduling Optimization

Department of Computer Science

Introdution

The restriction that we are placing on the input is that the weight of the orders is inversely proportional to the duration of the orders and also the jobs are sorted according to the earliest finish time.

Problem Statement

The algorithmic problem that we have planned to study is the weighted interval scheduling problem. In this version of the problem we are considering a single resource and multiple requests to it. These requests can be thought as requiring time corresponding to the resource. Each of these requests corresponds to an interval of time, S_i is the start time, F_i is the finish time, W_i is the weight of the request in which for any request $S_i < F_i$. For any two requests i and j to an resource are considered compatible i.e it can be carried out by the resource, if they don't overlap with each other. The overlapping condition can be like if $F_i \leq S_j$ then the requests are compatible. The goal is to find a valid subset of intervals with maximum total profit and are compatible to each other.

Algorithm for this problem

We are using Dynamic Programming and the recursive method to solve the weighted interval scheduling problem. The output is the schedule of requests that are non-overlapping and gives us the maximum profit. The greedy solution to this problem doesn't work because the requests have a weight parameter. The greedy algorithm looks to maximize the profit at each step and doesn't look ahead in terms of depth of the problem and cannot provide us the optimal solution. It works when all the requests are of unit weight. For example, if there are two jobs: the first has start time 1, finish time 3 and weight 1; the second has start time 2, finish time 4 and weight 100. The greedy algorithm schedules the first job, whereas the optimal one schedules the second.

The below illustrations shows the examples when greedy algorithm fails when the weights are not unity. It tries to schedule the intervals which has the earliest finish time or which has the maximum weight.

Suppose there are n orders. Each job i has a start time S_i, a finish time F_i, and a profit value W_i. We would like to compute a set T of compatible orders whose total weight is maximized.

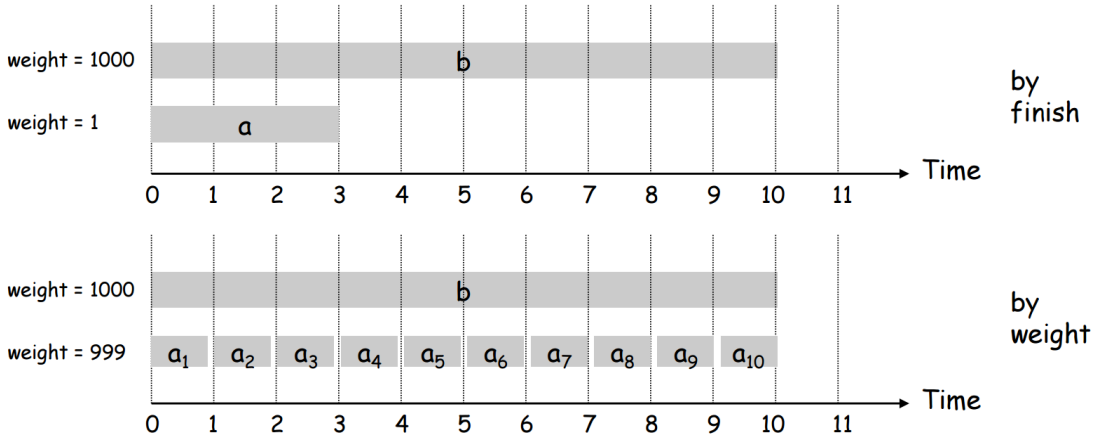

Figure 1: counter examples for greedy algorithm

The n jobs are are arranged in non-decreasing finish times F_i. Let $T(i)$ be the maximum weight of any set of compatible jobs, all of which finish by F_i. Define $m(i)$ to be the job that has the biggest index less than i which is compatible with job i and meets the condition of $f_j \leq s_i$.

The recurrence is $T(i) = max\{T(i-1), W(i) + T(m(i))\}$.

The algorithm first sorts the jobs by increasing finish times in case of no restriction. Compute the function $m(i)$ for i 1 to n. The initialization is done for the first value of the array as 0. In order to compute the actual schedule we use $T(i)$ to recover it or we store it in a separate array while we compute the maximum profit.

Execution using synthetic data

We generated a set of input jobs and stored them in a file. It was done in python and for every single job we generated a random number for the start time S_i, finish time F_i and the weight W_i. We started by giving an input of 1000 jobs initially and increased the number of jobs in steps of 500 till we reached 5500 jobs. For each execution of n, we gave two sets of inputs, one is with restriction and the one without restriction. The without restriction input was sorted according to the earliest finish time. Also the weights for each job in the two sets remained same for each job so that both the executions will give us the same maximum profit. The inputs were fed to a java program to calculate the optimal solution and running time.

Other Application

The weighted interval scheduling can also be applied in virtual machine scheduling in cloud computing. The intervals represent the a task or request that needs to be processed. We have a physical machine or server which has several virtual machine(VM) that can be allocated to process the requests. Each VM has a capacity that is a part of the capacity of the physical machine. Using interval scheduling we can find a optimal solution. We restricted the capacity of all the VMs to be allocated as same and is not being passed as an input. An example of the VM request is vm(0,6,2).

Results

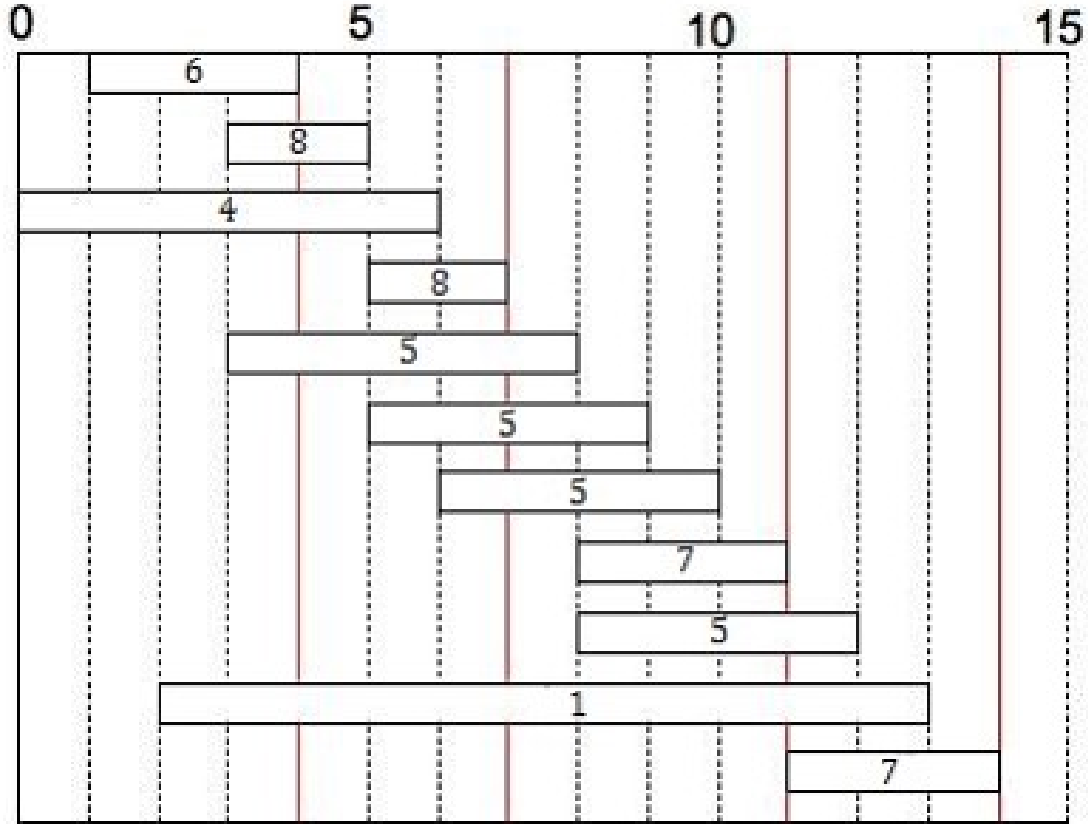

Figure 2: example of earliest finish time sorted graph

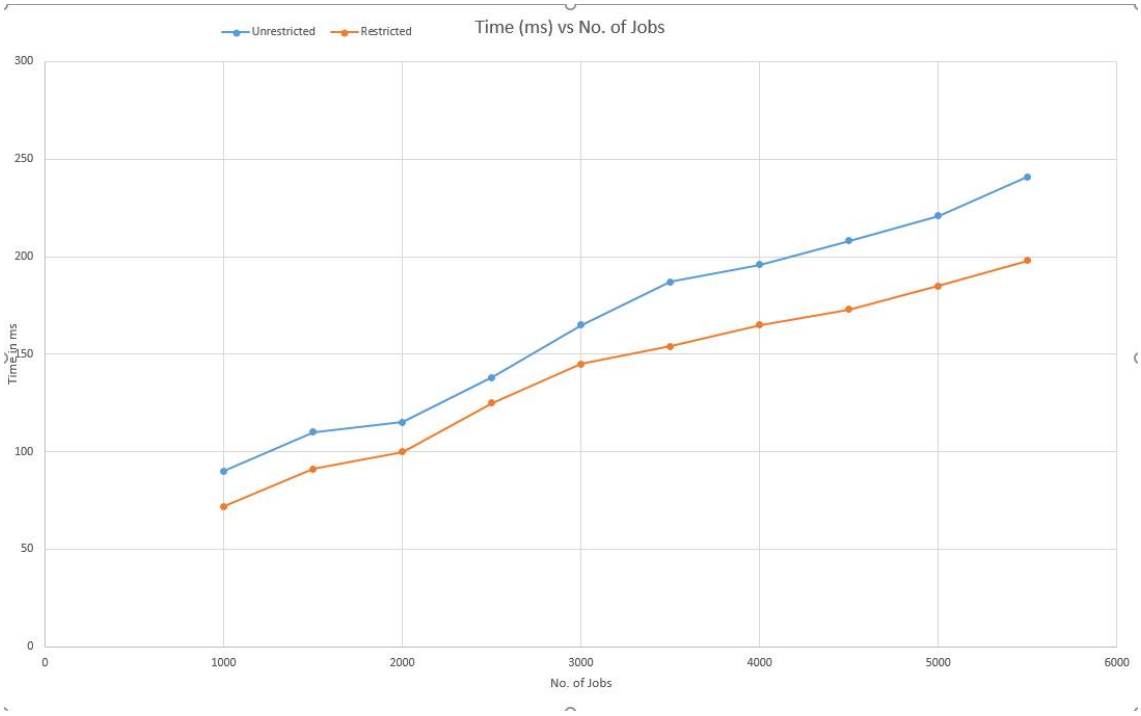

Figure 3: dynamic programming results

Virtual machine scheduling		
Jobs	VMs	Running time (m-s)
1000	300	232
5000	1500	510
10000	3000	932
20000	6000	1700
50000	18000	4900

The below graphs shows the time versus number of jobs comparison for a general case of single resource with multiple request and to find the optimal subset of intervals with maximum weight. We ran it with dynamic programming algorithm and with the recursive method.

Analysis on running time

This original version of the algorithm requires $O(n^2)$ since we need $O(n)$ time to solve each sub-problem.

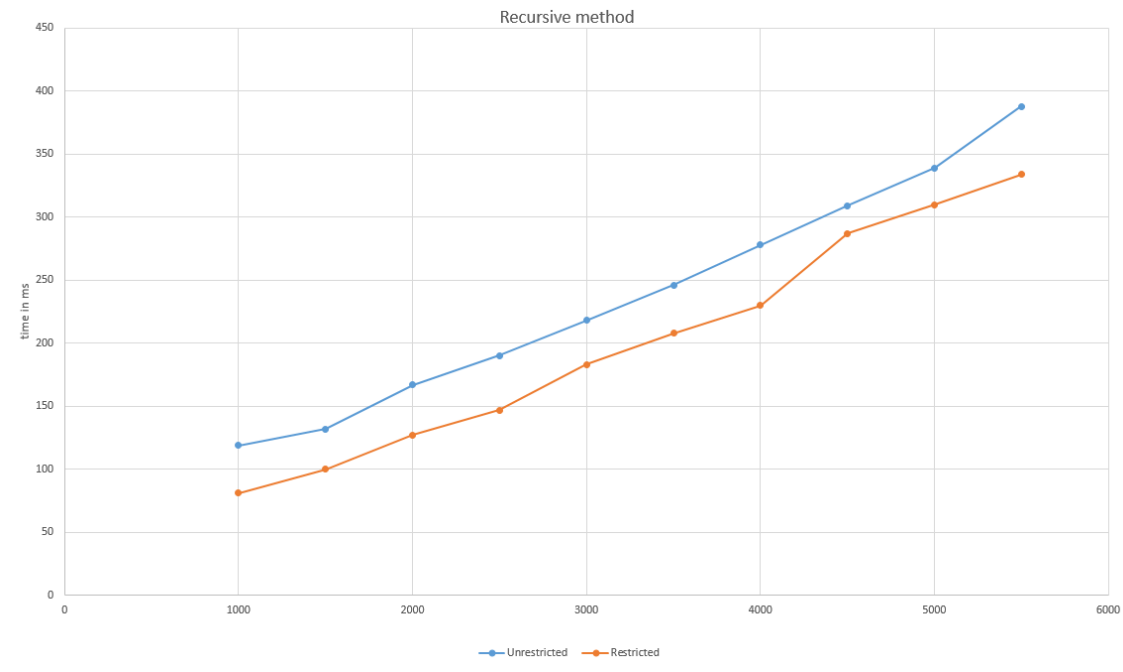

Figure 4: recursive method

References

[1] Erik Demaine, Srinivas Devadas, and Nancy Lynch. 6.046J Design and Analysis of Algorithms. Spring 2015. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

[2] Larry Ruzzo. CSE 521 Design and Analysis of Algorithms I. Winter 2013. University of Washington.

[3] Wenhong Dr. Tian, Yong Dr. Zhao. Optimized Cloud Resource Management and Scheduling.