Design and Analysis of Algorithms

L23: Job Scheduling

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT/KSSEM
rprustagi@ksit.edu.in

Resources

- Text book 2: Sec 4.1, 4.3, 4.4
- Text book 1: Sec 9.1-5.4 - Levitin
- R1: Introduction to Algorithms
  - Cormen et al.
- MIT Open Course Ware
  - https://www.geeksforgeeks.org/job-sequencing-using-disjoint-set-union/

Example Case

- In college fest which starts at 9:00am, there are a number of available events as below to participate, and each event takes `1` unit of time (e.g. 1hr).
  - Each event has different awards values
  - Each event has its own closing timeline.

Event	Closing	Award
`Mimcry`	12:00	200
`Drama`	11:00	100
`Painting`	12:00	90
`Dance`	10:00	50
`JAM`	11:00	125
`Singing`	10:00	60

Deadline
3
2
3
1
2
1

Q: What is the max award you can get?

Greedy Job Scheduling

- A set of n jobs to run on a computer
- Each job i has a deadline $d_i \geq 1$ and profit $p_i \geq 0$
- There is only one computer
- Each job takes one unit of time (simplification)
- Profit is earned when job is completed by deadline
- Find the subset of jobs that maximizes the profit, i.e.

  Maximize $\Sigma_{i \in J} P_i$

Note: It belongs to subset paradigm since we are looking at subset of jobs.

Example: Job Scheduling

Job	Profit	Dead -line
1	100	2
2	10	1
3	15	2
4	27	1

Feasible Solutions	Profit
1	100
2	10
3	15
4	27
1,2	110
1,3	115
1,4	127
2,3	25
3,4	42

Optimal Solution: 1,4

Job Scheduling: Greedy Approach

- What should be the optimization measure to schedule the next job?
- First attempt:
  - Choose $\Sigma_{i \in J} P_i$ as the optimization measure
  - i.e. choose a job that increases this value maximum
    - Subject to constraint of the deadline i.e. J (set of jobs) should be feasible solution.
  - How to choose jobs:
    - Order jobs in decreasing order of profit
    - Choose job one at a time as per this order and add to the solution if solution remains feasible.

Job Scheduling: Greedy Approach

Job	Pro fit	De ad-line
1	100	2
4	27	1
3	15	2
2	10	1

- Application of First Greedy approach
  - Job 1 is added to J. Feasible `{1}`
  - Next: Job 4 is considered as per order.
    - Is set `J={1,4}` feasible.
      - Yes if schedule is 4-1, No if 1-4
    - Thus `{1,4}` is feasible solution.
  - Next: Job 3 is considered,
    `{1,4,3}` is infeasible, thus `J` remains `{1,4}`
  - Next: Job 2 is considered
    `{1,4,2}` is infeasible thus `J` remains `{1,4}`
  - The max profit is `127` for `J={1,4}`
- Time complexity :
  - Evaluate feasibility for a given set: `n`!

Job Scheduling: Feasible Solution

- How to determine that a given set of jobs constitute feasible solution.
- Try out all possible permutations in jobs J
  - Check for each permutation if jobs can be scheduled meeting the deadlines.
- Easy to check for a given permutation $\sigma = i_1, i_2, \ldots, i_k$
  - Job i_q must be completed by time q, $1 \le q \le k$
  - If for some job i_q, $q > d_{i_q}$, then job i_q is not completed by d_{i_q}.
- When $|J| = k$, all $k!$ permutations must be checked
- Can we find one permutation that meets the need?
  - Order the jobs in non-decreasing order of deadlines

Proof for Feasible Solution

- Theorem 1:
  - Let J be the set of k jobs and $\sigma = i_1, i_2, \ldots, i_k$ is a permutation of jobs in J such that $d_{i_1} \le d_{i_2} \le \ldots \le d_{i_k}$.

    Then J is a feasible solution if and only if (*iff*) the jobs in J can be processed in the order σ without violating any deadline.
- Theorem 2:
  - The greedy method (order jobs in non-increasing order of profit) always obtains an optimal solution to the job scheduling problem.

Algo High Level

Algo `GreedyJob`(int `d[]`, set `J`, int `n`) `{`

```
// J is set of jobs that can be completed in deadlines d[]
J={1}
for i=2 to n {
    if all jobs in J ∪ {i} can be completed, then
        // by their deadlines
        J = J ∪ {i}
}
}
```

Algo-1: Job Scheduling

int `JobSchedule2`(*int* `d[]`, *int* `j[]`, *int* `n`) {

//$n \geq 1$, and deadlines $d[i] \geq 1$, $1 \leq i \leq n$

//Jobs are ordered such that their profits are in non-increasing order i.e. $p[1] \geq p[2] \geq \ldots \geq p[n]$.

//$J[i]$ is the i^{th} job in the optimal solution with $k \leq n$ jobs

// At algo termination, $d[J[i]] \leq d[j[i+1]]$, $1 \leq i < k$

// Initialize

`d[0] = 0` // fictitious job with deadline of 0

// allows for job insertion at position 1 later.

`J[0] = 0` // this job is boundary and can't be scheduled

`J[1] = 1` // start with job 1 with highest profit

`k = 1` // job set size is 1 to start with

Algo1: Job Scheduling

```
for(i=2; i≤n; i++) {
  // consider jobs in non-increasing order of p[i]
  // find pos for J[i] and check for feasibility of insertion
  int r = k  //job set size
  while ((d[J[r]]>d[i]) &&(d[J[r]!=r))
      r–; //find position where job i can be considered.
  if ((d[J[r]]≤d[i]) &&(d[i]>r)){
      //insert i into J[]
      for (int q=k; q≥(r+1); q--)
          J[q+1]=J[q] // increase deadline of jobs by 1.
      J[r+1]=i
      k++ // since job i is feasible, increase the set size.
  }//end if
}//end for i
return k
}
```

Algo-1:Time Complexity

- For loop run n times.
  - Each job needs to be considered.
- if K is the value of max deadline, then
  - Inside while loop plus for loop (for shifting slots) may run K times.
- Time complexity: $O(nK)$
- Considering K is of order of n (if all jobs can be scheduled)
- Time complexity: $O(n^2)$

Algo-2: Job Scheduling

//Approach: schedule a job in the slot where it meets deadline.
// If no slot is available before deadline, then job is not scheduled.
// jobs are ordered in non-increasing order as per deadlines.

```
int JobSchedule-1(int d[], int j[], int n) {
  //n≥1, and deadlines d[i]≥1, 1≤i≤n
  //Jobs are ordered such that their profits are in non-
  increasing order i.e. p[1]≥p[2]≥…≥p[n] .
  //Job[i] is ith job in the optimal solution with k≤n jobs
  // At algo termination, d[Job[i]]≤d[job[i+1]],
  1≤i<k
  // Initialization
  k=0;  // size of Job schedule
  for i=1 to n
     slot[i]=False // all slots are initialized to false
```

Algo-2: Job Scheduling

```
for (i=1; i≤n; i++) {
  // consider jobs in non-increasing order of p[i]
  //check if any slot available before deadline
  while (j=d[i]; j>0; j—) {
      //find position where job i can be considered.
      if (slot[j] == False{
        //Add jobs to the slot
        slot[j] = True;
        Job[j] = i;
        k++;
        break;  // from while
      }//end if
  }//end while
} // end for
return  k
}
```

Algo-2: Time Complexity

- For loop run n times.
  - Each job needs to be considered.
- if `K` is the value of max deadline, then
  - while loop may run `K` times.
- Time complexity: $O(nK)$
- Considering K is of order of n (if all jobs can be scheduled)
- Time complexity: $O(n^2)$

Fast Job Scheduling (Union-Find)

- Let `i` denote the timeslot `i`
  - At the start time, each time slot is its own set
- There are m timeslots, where
  $m = \min(n, \max(d_i))$
  - i.e. the latest deadline
- Each set of `k` slots has a value `F(k)` for all slots `i` set `k`
  - `F(k)` : Stores highest free timeslot before this time
  - `F(k)` : Defined only for root node in set
- Initially all slots are free

Summary

- Job Scheduling
  - Greedy approach: Schedule as per profit and deadline
- Two approaches
  - Schedule the job in earliest slot and then keep shifting right
  - Schedule the job in the deadline slot or look for slots earlier than the deadline