Course Name: Analysis and Design of Algorithms

Topics to be covered

- Algorithms
  - What is an Algorithm?
  - Characteristics
  - Complexity

JOB SEQUENCING WITH DEADLINES

The problem is stated as below.

- There are n jobs to be processed on a machine.
- Each job i has a deadline $d_i \geq 0$ and profit $p_i \geq 0$.
- Pi is earned iff the job is completed by its deadline.
- The job is completed if it is processed on a machine for unit time.
- Only one machine is available for processing jobs.

JOB SEQUENCING WITH DEADLINES (Contd..)

- A feasible solution is a subset of jobs J such that each job is completed by its deadline.
- An optimal solution is a feasible solution with maximum profit value.

Example : Let n = 4, $(p_1,p_2,p_3,p_4) = (100,10,15,27)$, $(d_1,d_2,d_3,d_4) = (2,1,2,1)$

JOB SEQUENCING WITH DEADLINES (Contd..)

Sr.No.	Feasible Solution	Processing Sequence	Profit value
(i)	(1,2)	(2,1)	110
(ii)	(1,3)	(1,3) or (3,1)	115
(iii)	(1,4)	(4,1)	127 is the optimal one
(iv)	(2,3)	(2,3)	25
(v)	(3,4)	(4,3)	42
(vi)	(1)	(1)	100
(vii)	(2)	(2)	10
(viii)	(3)	(3)	15
(ix)	(4)	(4)	27

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION

- Consider the jobs in the non increasing order of profits subject to the constraint that the resulting job sequence J is a feasible solution.
- In the example considered before, the non-increasing profit vector is

  (100 27 15 10) (2 1 2 1) p_1 p_4 p_3 p_2
  d_1 d_4 d_3 d_2

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

J = { 1} is a feasible one

J = { 1, 4} is a feasible one with processing sequence (4,1)

J = { 1, 3, 4} is not feasible

J = { 1, 2, 4} is not feasible

J = { 1, 4} is optimal

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

Theorem: Let J be a set of K jobs and

$\sigma = (i_1, i_2, \ldots. i_k)$ be a permutation of jobs in J such that $di_1 \leq di_2 \leq \ldots \leq di_k$.

- J is a feasible solution iff the jobs in J can be processed in the order σ without violating any deadly.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

Proof:

- By definition of the feasible solution if the jobs in J can be processed in the order without violating any deadline then J is a feasible solution.
- So, we have only to prove that if J is a feasible one, then σ represents a possible order in which the jobs may be processed.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- Suppose J is a feasible solution. Then there exists $\sigma^1 = (r_1, r_2, \ldots, r_k)$ such that

  $d_{rj} \geq j, \qquad 1 \leq j < k$

  i.e. $d_{r1} \geq 1$, $d_{r2} \geq 2$, …, $d_{rk} \geq k$.

  each job requiring an unit time.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- $\sigma = (i_1, i_2, \ldots, i_k)$ and $\sigma^1 = (r_1, r_2, \ldots, r_k)$
- Assume $\sigma^1 \neq \sigma$. Then let a be the least index in which σ^1 and σ differ. i.e. a is such that $r_a \neq i_a$.
- Let $r_b = i_a$, so b > a (because for all indices j less than a $r_j = i_j$).
- In σ^1 interchange r_a and r_b.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

$\sigma = (i_1, i_2, \ldots\ i_a \quad i_b \quad i_k)$ [r_b occurs before r_a in $i_1, i_2, \ldots, i_k$]

$\sigma^1 = (r_1, r_2, \ldots\ r_a \quad r_b \ \ldots \ r_k)$

$i_1 = r_1,\ i_2 = r_2, \ldots i_{a-1} = r_{a-1},\ i_a \neq r_b$ but $i_a = r_b$

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- We know $di_1 \leq di_2 \leq \ldots di_a \leq di_b \leq \ldots \leq di_k$.
- Since $i_a = r_b$, $dr_b \leq dr_a$ or $dr_a \geq dr_b$.
- In the feasible solution $dr_a \geq a$ $dr_b \geq b$
- So if we interchange r_a and r_b, the resulting permutation $\sigma^{11} = (s_1, \ldots s_k)$ represents an order with the least index in which σ^{11} and σ differ is incremented by one.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- Also the jobs in σ^{11} may be processed without violating a deadline.
- Continuing in this way, σ^{1} can be transformed into σ without violating any deadline.
- Hence the theorem is proved.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- **Theorem2:**The Greedy method obtains an optimal solution to the job sequencing problem.
- Proof: Let(p_i, d_i) $1 \le i \le n$ define any instance of the job sequencing problem.
- Let I be the set of jobs selected by the greedy method.
- Let J be the set of jobs in an optimal solution.
- Let us assume $I \neq J$.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- If J C I then J cannot be optimal, because less number of jobs gives less profit which is not true for optimal solution.
- Also, I C J is ruled out by the nature of the Greedy method. (Greedy method selects jobs (i) according to maximum profit order and (ii) All jobs that can be finished before dead line are included).

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- So, there exists jobs a and b such that $a \in I$, $a \notin J$, $b \in J, b \notin I$.
- Let a be a highest profit job such that $a \in I$, $a \notin J$.
- It follows from the greedy method that $p_a \geq p_b$ for all jobs $b \in J, b \notin I$. (If $p_b > p_a$ then the Greedy method would consider job b before job a and include it in I).

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- Let S_i and S_j be feasible schedules for job sets I and J respectively.
- Let i be a job such that $i \in I$ and $i \in J$.

  (i.e. i is a job that belongs to the schedules generated by the Greedy method and optimal solution).
- Let i be scheduled from t to t+1 in S_I and $t^1$to t^1+1 in S_j.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- If $t < t^1$, we may interchange the job scheduled in $[t^1\ t^1+1]$ in S_I with i; if no job is scheduled in $[t^1\ t^1+1]$ in S_I then i is moved to that interval.
- With this, i will be scheduled at the same time in S_I and S_J.
- The resulting schedule is also feasible.
- If $t^1 < t$, then a similar transformation may be made in S_j.
- In this way, we can obtain schedules S_I^1 and S_J^1 with the property that all the jobs common to I and J are scheduled at the same time.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- Consider the interval [T_a T_a+1] in S_I^1 in which the job a is scheduled.
- Let b be the job scheduled in S_j^1 in this interval.
- As a is the highest profit job, $p_a \geq p_b$.
- Scheduling job **a** from t_a to t_a+1 in S_j^1 and discarding job b gives us a feasible schedule for job set J^1 = J-{b} U {a}. Clearly J^1 has a profit value no less than that of J and differs from in one less job than does J.

GREEDY ALGORITHM TO OBTAIN AN OPTIMAL SOLUTION (Contd..)

- i.e., J^1 and I differ by m-1 jobs if J and I differ from m jobs.
- By repeatedly using the transformation, J can be transformed into I with no decrease in profit value.
- Hence I must also be optimal.

GREEDY ALGORITHM FOR JOB SEQUENSING WITH DEADLINE

Procedure greedy job (D, J, n) J may be represented by

// J is the set of n jobs to be completed// one dimensional array J (1: K)

// by their deadlines // The deadlines are

J ← {1} D (J(1)) ≤ D(J(2)) ≤ .. ≤ D(J(K))

for I ← 2 to n do To test if JU {i} is feasible,

If all jobs in JU{i} can be completed we insert i into J and verify

by their deadlines D(J®) ≤ r 1 ≤ r ≤ k+1

then J ← JU{I}

end if

repeat

end greedy-job

GREEDY ALGORITHM FOR SEQUENCING UNIT TIME JOBS

Procedure JS(D,J,n,k)

// D(i) $\geq$ 1, 1$\leq$ i $\leq$ n are the deadlines //

// the jobs are ordered such that //

// $p_1 \geq p_2 \geq \ldots\ldots \geq p_n$ //

// in the optimal solution ,D(J(i) $\geq$ D(J(i+1)) //

// 1 $\leq$ i $\leq$ k //

integer D(o:n), J(o:n), i, k, n, r

D(0) $\leftarrow$J(0) $\leftarrow$ 0

// J(0) is a fictious job with D(0) = 0 //

K$\leftarrow$1; J(1) $\leftarrow$1 // job one is inserted into J //

for i $\leftarrow$2 to do // consider jobs in non increasing order of pi //

GREEDY ALGORITHM FOR SEQUENCING UNIT TIME JOBS (Contd..)

// find the position of i and check feasibility of insertion //

r← k // r and k are indices for existing job in J //

// find r such that i can be inserted after r //

while D(J(r)) > D(i) and D(i) ≠ r do

// job r can be processed after i and //

// deadline of job r is not exactly r //

r← r-1 // consider whether job r-1 can be processed after i //

repeat

GREEDY ALGORITHM FOR SEQUENCING UNIT TIME JOBS (Contd..)

if D(J(r)) ≥ d(i) and D(i) > r then

// the new job i can come after existing job r; insert i into J at position r+1 //

for I ← k to r+1 by –1 do

J(I+1)← J(l) // shift jobs(r+1) to k right by//

//one position //

repeat

GREEDY ALGORITHM FOR SEQUENCING UNIT TIME JOBS (Contd..)

J(r+1)←i ; k ←k+1

// i is inserted at position r+1 //

// and total jobs in J are increased by one //

repeat

end JS

COMPLEXITY ANALYSIS OF JS ALGORITHM

- Let n be the number of jobs and s be the number of jobs included in the solution.
- The loop between lines 4-15 (the for-loop) is iterated (n-1)times.
- Each iteration takes O(k) where k is the number of existing jobs.

∴The time needed by the algorithm is 0(sn) $s \leq n$ so the worst case time is $0(n^2)$.

If $d_i = n - i+1 \quad 1 \leq i \leq n$, JS takes $\theta(n^2)$ time

D and J need $\theta(s)$ amount of space.

A FASTER IMPLEMENTATION OF JS

- The time of JS can be reduced from $0(n^2)$ to $0(n)$ by using SET UNION and FIND algorithms and using a better method to determine the feasibility of a partial solution.
- If J is a feasible subset of jobs, we can determine the processing time for each of the jobs using the following rule.

A FASTER IMPLEMENTATION OF JS (Contd..)

- If job I has not been assigned a processing time, then assign it to slot [α -1, α] where α is the largest integer r such that $1 \leq r \leq d_i$ and the slot [α -1, α] is free.
- This rule delays the processing of jobs i as much as possible, without need to move the existing jobs in order to accommodate the new job.
- If there is no α, the new job is not included.

A FASTER IMPLEMENTATION OF JS (Contd..)

EXAMPLE: let n = 5, (p_1,--------p_5) = (20,15,10,5,1) and (d_1,--d_5) = (2,2,1,3,3). Using the above rule

J or	assigned slot	jobs being considered	action
Ø	none	1	assigned to [1, 2]
{1}	[1,2]	2	[0,1]
{1,2}	[0,1],[1,2]	3	cannot fit reject as [0,1] is not free
{1,2}	[0,1],[1,2]	4	assign to [2,3]
{1,2,4}	[0,1],[1,2],[2,3]	5	reject

The optimal solution is {1,2,4}

A FASTER IMPLEMENTATION OF JS (Contd..)

- As there are only n jobs and each job takes one unit of time, it is necessary to consider the time slots [i-1,i] $1 \leq i \leq b$ where b = min {n, max $\{d_i\}$}
- The time slots are partitioned into b sets .
- i represents the time slot [i-1,i]

  [0 1]is slot 1

  [1 2] is slot 2
- For any slot i, n_i represents the largest integers such that $n_i \leq i$ and slot n_i is free.

  If [1,2] is free

  n_2=2 otherwise

  n_2=1 if [0 1] is free

A FASTER IMPLEMENTATION OF JS (Contd..)

- To avoid end condition, we introduce a fictious slot [-1, 0] which is always free.
- Two slots are in the same set iff $n_i = n_j$
- If i and j i < j are in the same set, then i, i+1, i+2,…i are in the same set.
- Each set k of slots has a value f(k) ,$f(k)=n_i$, for all slots i in set k. (f(k) is the root of the tree containing the set of slots k)
- Each set will be represented as a tree.

A FASTER IMPLEMENTATION OF JS (Contd..)

- Initially all slots are free and $f(i) = i$ $1 \leq i \leq b$.
- $P(i)$ represents as a negative number the number of nodes in the tree represented by the set with slot I.
- $P(i) = -1$ $0 \leq i \leq b$ initially.
- If a job with deadline d is to be scheduled, we find the root of the tree containing the slot min {n, d}.

A FASTER IMPLEMENTATION OF JS (Contd..)

EXAMPLE: For the problem n =5 ($p_1 \ldots p_5$) = (20,15,10,5,1), (d_1------d_5) = (2,2,1,3,3) the trees defined by the P(i)'s are

J 1 2 3 4 5 Job considered

-1 -1 -1 -1 -1 - 1 1,d_1 = 2 [1,2] is free

ϕ P(0) P(1) P(2) P(3) P(4) P(5)

1 0 1 P(1) F(1)=0 2, d_1 = 2

-2 F(1) = 1 [0,1] free

P(0) P(2) {1,2} -3 P(1)2 F[1] = 0 reject

0 0

A FASTER IMPLEMENTATION OF JS (Contd..)

The algorithm for fast job scheduling (FJS) is as follows

Procedure FJS (D,n,b,j,k)

// Find an optimal solution J = J(1),…,J(k) $1 \le k \le n$ //

// It is assumed that $p_1 \ge p_2 \ge \ldots p_n$ and b = min{n, max(d(i)) //

Integer b, D(n) ,J(n),F(O: b), P(O: b)

for i← 0 to b. Do // initialize trees //

F(i)← i; P(i)←-1

repeat

K ←0 // Initialize J //

A FASTER IMPLEMENTATION OF JS (Contd..)

For i ← 1 to n do // use greedy rules //

j ← FIND (min (n, D(i)) // F(j) is the nearest free slot if F(j) ≠ 0 //

if F(j) ≠ 0 then k ← k+1 ; J(k) ← i

All slots are not occupied

//select job i //

L ← Find (F(j)-1); call union (L, j)

F(j) ← F(L) // j may be new root //

endif

repeat

end FJS

A FASTER IMPLEMENTATION OF JS (Contd..)

It is F(j) –1 because you need to union J with I which

is F(j) -1.

F(i) is a value for a set of slots with I which is F(j)-1

F(k) = n_i for all slots in the set k.

n_i is that largest integer such that

$n_i \leq i$ and slot n_i is free

F(1)=1 [0 1]

F(2)=2 [1 2]

P(i)= is the number of nodes in the tree respectively the

set with slot

A FASTER IMPLEMENTATION OF JS (Contd..)

Complexity of algorithm FJS

As there are n unions and 2n finds, in the for loop the computing time is

$0(n \alpha(2n , n))$

α (m, n)   m $\geq$ n is related to Ackermal function

α (m,n)= min $\{z \geq 1/A(3, 4[m/n]) > logn_2\}$

For all practiced purposes, we may

assume log n < A(3,4) and hence

α (m,n) $\leq$ 3   m $\geq$ n

$\therefore$ The computing time of FJS is O(n)

Additional 2n words of space for F and P are required.