CS 1510
Algorithm Design

Greedy Algorithms & Dynamic Programming

Problems 12, 13 and 1

Due Monday September 8, 2014

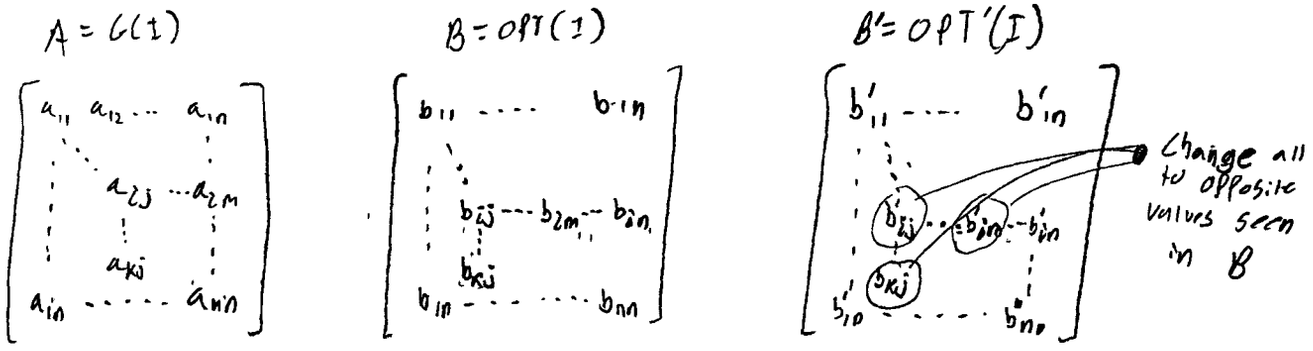
Buck Young and Rob Brown

Problem 12

Input: Positive integers $r_1, r_2, ..., r_n$ and $c_1, c_2, ..., c_n$
Output: An n by n matrix with 0/1 entries such that for all i the sum of the i^{th} row in a is r_i and the sum of the i^{th} column is c_i.
Theorem: The greedy algorithm (G) given in problem 12 is correct.

Proof: Suppose for the purpose of reaching a contradiction that there exists some input I consisting of two sets $\{r_1, r_2, ..., r_n\}$ and $\{c_1, c_2, ..., c_n\}$ such that G produces unacceptable output. Let $G(I) = A$ be the matrix that G produces, and let $OPT(I) = B$ be the matrix that some optimal algorithm which agrees for the most number of steps produces.

Consider the first pair of indices (i, j) where the indexed values of A disagree with those of B. In other words, consider (i, j) such that $\forall (x, y) \leq (i, j) : a_{ij} = b_{ij}$.



Since $a_{ij} = b_{ij}$, and both A and B consist entirely of 0/1 values, we have two cases:

1) $a_{i,j} = 0$ and $b_{i,j} = 1$
2) $a_{i,j} = 1$ and $b_{i,j} = 0$

Consider case 1. Since $b_{i,j} = 1$ and $(\sum_{w=1}^{n} b_{i,w}) = r_i$ we know that in some column (m) of B's i^{th} row there exists $b_{i,m} = 0$ where $b_{i,m} \neq a_{i,m}$ (ie, $a_{i,m} = 1$). A is constructed so that the a_i^{th} row has exactly r_i 1's. If OPT gives B a 1 at (i, j), it must compensate later by NOT having a 0 at some later point. This is the value we call $b_{i,m}$. Likewise, since OPT must produce $(\sum_{h=1}^{n} b_{h,j}) = c_j$, there must be some $b_{k,j}$ in the j^{th} column of B such that $b_{k,j} = 0$ and $a_{k,j} = 1$.

By the same logic, for case 2 we can deduce that there is some column (m) in B's i^{th} row where $b_{i,m} = 1$ and $a_{i,m} = 0$; likewise, there is some row k in B's j^{th} column where $b_{k,j} = 1$ and $a_{k,j} = 0$.

Let OPT'(I) = OPT(I) = B with the values of $b_{i,j}$, $b_{i,m}$, and $b_{k,j}$ changed to their opposite values.

CASE 1

$$\sum_{w=1}^{n} b'_{i,w} = r_i = r_i - b_{i,j} - b_{i,m} + !(b_{i,m}) + !(b_{i,j}) = r_i - 1 - 0 + 0 + 1 = r_i$$

$$\sum_{h=1}^{n} b'_{h,j} = c_j = c_j - b_{i,j} - b_{k,j} + !(b_{k,j}) + !(b_{i,j}) = c_j - 1 - 0 + 0 + 1 = c_j$$

CASE 2

$$\sum_{w=1}^{n} b'_{i,w} = r_i = r_i - b_{i,j} - b_{i,m} + !(b_{i,m}) + !(b_{i,j}) = r_i - 0 - 1 + 1 + 0 = r_i$$

$$\sum_{h=1}^{n} b'_{h,j} = c_j = c_j - b_{i,j} - b_{k,j} + !(b_{k,j}) + !(b_{i,j}) = c_j - 0 - 1 + 1 + 0 = c_j$$

We have thus demonstrated that, if OPT was indeed correct to begin with (ie, respective sums equaled c_j ad r_i), these modifications do not effect the correctness of OPT'. It is also evident that OPT' is more like G for one additional step.

$\therefore$ We have reached a contradiction because OPT' agrees with G for one additional step despite OPT being defined as agreeing for the most number of steps.

Problem 13

(a)

Input: A set of n jobs j – each job is described by an integer release time and deadline (r_j, d_j) – each of which must run for one unit of time. Additionally at each time interval t $\in [0, T)$, the machine may be ON or OFF at the time interval $(t, t+1)$. Let T equal the maximum deadline – $\max(d_j)$ for all j.

Output: Whether or not a feasible sequence was found in which every job ran for one unit of time at some interval after r_j and before d_j.

Algorithm: Run the job with the earliest deadline first:

while $t < T$ **do**
  if the machine is on at time t **then**
    for each job that has not yet run && is released – $(r_j \leq t)$ **do**
      Run the job with the earliest deadline – $(\min(d_j)$ for all $j)$ – (break ties arbitrarily)

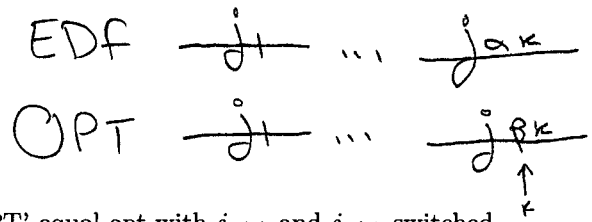
if all jobs in the input I have run by time T **then**
  return FEASIBLE
else
  return NOT FEASIBLE

Theorem: The described "earliest deadline first" algorithm EDF is correct.

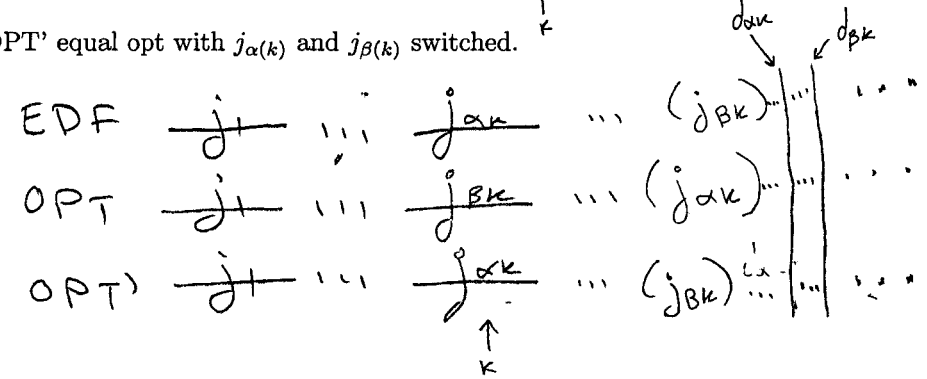
Proof: Assume to reach a contradiction that there exists an input on which EDF produces an unacceptable output.

Let EDF(I) = $\{j_{\alpha(1)}, ..., j_{\alpha(n)}\}$ be the output of EDF on input I
& OPT(I) = $\{j_{\beta(1)}, ..., j_{\beta(n)}\}$ be the optimal solution which agrees with EDF(I) for the most number of steps

Let k be the first point of disagreement between EDF and OPT



Let OPT' equal opt with $j_{\alpha(k)}$ and $j_{\beta(k)}$ switched.



Clearly OPT' is more like EDF. In order to prove that it still produces an acceptable output, consider the following four cases. Also note that by the definition of EDF, we know that the deadline $d_{\alpha(k)}$ is earlier

than (or tied with) the deadline $d_{\beta(k)}$. Stated another way: $d_{\alpha(k)} \leq d_{\beta(k)}$

Case 1: Both EDF and OPT determined the solution was FEASIBLE

Since we know $d_{\alpha(k)} \leq d_{\beta(k)}$, we know that $d_{\beta(k)}$ is after the scheduled $j_{\alpha(k)}$ in OPT. Therefore we can easily switch $j_{\alpha(k)}$ and $j_{\beta(k)}$ and both jobs will complete before their deadlines. Therefore an acceptable solution is still produced.

Case 2: Both EDF and OPT determined the solution was NOT FEASIBLE

Since a NOT FEASIBLE solution was determined, we know that some job was unable to be scheduled before its deadline. Therefore switching $j_{\beta(k)}$ and $j_{\alpha(k)}$ (or scheduling one and not the other) should have no affect on the final outcome and an acceptable solution is still produced.

Case 3: EDF determined FEASIBLE but OPT determined NOT FEASIBLE

3

This state is impossible by the definition of OPT. If EDF was able to create a FEASIBLE solution, then OPT is guaranteed to also produce a FEASIBLE solution, by definition.

Case 4: EDF determined NOT FEASIBLE but OPT determined FEASIBLE

In this case, we can do the switch using what we know about the deadlines ($d_{\alpha(k)} \leq d_{\beta(k)}$) and OPT' still produces an acceptable solution.

All in all, OPT' always produces an acceptable solution.

$\therefore$ We have reached a contradiction because OPT' is more like EDF for one more step even though OPT was defined as being most like EDF.

Problem 13

(b)

Input: A set of n jobs j – each job is described by an integer release time and deadline (r_j, d_j) – each of which must run for one unit of time. Let T equal the maximum deadline – $\max(d_j)$ for all j. Additionally, a positive integer L is given which defines the length of an ON state.

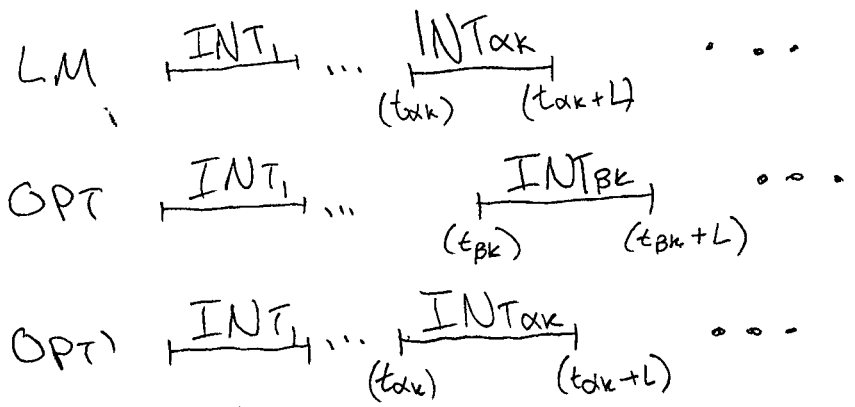
Output: The minimum number k times that the machine must turn ON in order to feasibly schedule all jobs within the intervals of length L.

Algorithm: While there are schedulable jobs in the input set, place an interval of length L at the left-most point t where that interval $(t + L)$ contains the most schedulable jobs $\leq L$. Then schedule jobs within that interval according to the earliest deadline first. } Not correct
(Note that a schedulable job is one that has been released and is not past it's deadline.)

Theorem: The described "left most" algorithm LM is correct.

Proof: Assume to reach a contradiction that there exists an input I on which LM produces and unacceptable output.

Let INT_1, ..., INT_n be the created intervals of length L starting at t_1, ..., t_n
& LM(I) = $\{INT_{\alpha(1)}, ..., INT_{\alpha(n)}\}$ be the output of LM on input I
& OPT(I)= $\{INT_{\beta(1)}, ..., INT_{\beta(n)}\}$ be the optimal solution which agrees with LM the most number of steps
& k be the first point of disagreement between LM and OPT



Note: we know that $t_{\alpha(k)} < t_{\beta(k)}$, by the definition of LM
This is tricky because it also chooses interval based on most # of jobs

Let OPT' equal OPT with $INT_{\beta k}$ shifted to the left such that it starts at $t_{\alpha(k)}$ and is, of course, of length L – and therefore becomes (indistinguishable from) $INT_{\alpha(k)}$.

Clearly OPT' is more like LM for one additional interval.

A few observations about the shift:
1) OPT clearly hasn't missed scheduling a job, by the definition of OPT (therefore the space from $t_{\alpha(k)}$ to $t_{\beta(k)}$ is void of a completely unscheduled job in OPT) } This is not clear since the interval changed.

2) OPT has not created an interval that scheduled more jobs than $INT_{\alpha(k)}$, by the definition of LM

Thus, shifting $INT_{\beta(k)}$ to the left so that it becomes $INT_{\alpha(k)}$ in OPT' will still produce an acceptable output.

$\therefore$ We have reached a contradiction because OPT' agrees with LM for one additional step despite OPT being defined as agreeing with LM for the most number of steps.

Problem 1

(a)

```
procedure T(int n)
    if n < 0 then
        return -1
    if n == 0 or n==1 then
        return 2;
    SUM = 0
    for 1 ≤ i ≤ n − 1 do
        SUM+=T(i)*T(i-1)
    return SUM
```

Not true.

Recursive algorithms which only reduce the input by a constant size each call are inherently exponential. Here, we have an algorithm which does this (ie, only decreases the input size by a constant size each call), and additionally creates this many more recursive calls at each point in the recursion. Clearly, it is exponential.

(b)

```
procedure T(int n)
    if n < 0 then
        return -1
    if n == 0 or n==1 then
        return 2;
    T[0] = 2, T[1] = 2
    for 2 ≤ i ≤ n do
        SUM = 0
        for 1 ≤ j ≤ n − 1 do
            SUM+=T[j]*T[j-1]
        T[i] = SUM
    return T[n]
```

Need english descriptions of algorithms as well

Clearly this algorithm is $\mathcal{O}(n^2)$. This is achieved by starting at the base ($n = 0, n = 1$) and computing the the next value based on the sum of all the previous; as opposed to starting at the top and recomputing everything at each step closer to the base.

(c)

```
procedure T(int n)
    if n < 0 then
        return -1
    if n == 0 or n==1 then
        return 2;
    T[0] = 2, T[1] = 2, T[2] = 4
    for 3 ≤ i ≤ n do
        T[i] = T[i-1]*T[i-2] + T[i-1]
    return T[n]
```

Clearly this algorithm is $\mathcal{O}(n)$.