Applied Algorithms

Prof. Tami Tamir

Review Q&A

Scheduling 1

n jobs need to be processed by two machines.

M_1 can process any number of jobs simultaneously.

M_2 can only process one job at a time.

Job j needs to be processed a_j and b_j time units on M_1, M_2 respectively.

Suggest an algorithm for minimizing the makespan of the schedule.

1. When each job must be processed by M_1 before it is processed by M_2.
2. When each job must be processed by M_2 before it is processed by M_1.

Solution 1:

All the jobs will be assigned to M_1 at time 0. Job j will be available to M_2 at time aj. Therefore, the problem is equivalent to $1|r_j|C_{max}$ (minimum makespan on a single machine with release times).

The makespan of M_2 equals $\Sigma_j b_j$ + idle time of M_2. The first term is a constant. In order to minimize the idle time of M_2 we should 'feed' it jobs whenever possible. This is achieved by schedule with no intended idles, in particular, by scheduling the jobs on M_2 in order $a_1 \leq a_2 \leq \ldots \leq a_n$.

Proof: Assume that the order is different, that is, there is a schedule S with a pair of two jobs k,j, such that k is scheduled on M_2 just before job j but $a_k > a_j$. Consider the schedule S' in which these two jobs are swapped.

Since $a_j < a_k$, job j was available to M_2 before job k. In particular, in S, M_2 cannot be idle between k and j. In S' the completion time of k is therefore at most the completion time of j in S. Therefore the idle time of M_2 could only decrease by the swapping (jobs scheduled after the pair might be scheduled earlier).

Solution 2:

The completion time of job j is $C_j + a_j$ where C_j denotes the completion time of the job on M_2. Specifically $C_j = b_1 + b_2 + \ldots + b_j$. The goal is to minimize $\max_j C_j + a_j$.

Claim: This term is minimized when the jobs are processed on M_2 such that $a_1 \geq a_2 \geq \ldots \geq a_n$.

Proof idea: We would like to match large a_j value with small C_j value. The a_j's are given as input; the Cj's are determined by our algorithm.

More formal proof using exchange argument: Consider a schedule S in which for some pair of jobs, $a_k < a_j$ but J_k is processed just before J_j. Let S' be the schedule in which they are swapped. The completion time of the other jobs is not affected. The completion time of a_j is reduced; the completion time of a_k is increased by b_j. However, $C_k(S') = C_j(S)$, therefore when adding the a's, we get that the value of $\max_j C_j + a_j$ could only decrease => S' is not worse than S. (This is similar to the optimality of proof of EDD for $1||L_{max}$).

Scheduling 2:

Let I be an instance of n=20 jobs for which an optimal solution of $P2||C_{max}$ has value 100.

Let I' be an instance derived from I by increased by 1 the processing times of all 20 jobs.

Prove or give a counter example:

1. The solution of $P2||C_{max}$ for I' has value at least 110.
2. The solution of $P2||C_{max}$ for I' has value at most 110.
3. The solution of $P2||C_{max}$ for I' has value at most 119.

Scheduling 2, solution:

a. False, Take I={100,19 *1}.

b. False, Take I={100, 82, 18*1}. We get I’={101, 83, 18*2}. The total size of jobs is 220. Note that any combination of 101 with short jobs has an odd-value total size, therefore it will never sum up to 110.

c. True. In the optimal solution for I, there are at most 19 jobs on a single machine, therefore, by keeping the same assignment of jobs to machines, we get a solution for I’ whose makespan is extended by at most 19.

Scheduling 3:

Given an instance of $F2||C_{max}$, (minimum makespan on two machines, flowshop scheduling), consisting of n jobs. It is known that for all j, $a_j=b_j=2^j$. One of the two machines must be replaced by a slower machine, in which the processing time of every task is doubled. You can select which of the two machines will be replaced. Does it matter? If yes, which machine do you prefer to slow down such that the makespan will be hurt the least? If not, explain why?

Scheduling 3:

It doesn't matter.
If the first machine is hurt, then for all jobs $a_j > b_j$ and by Johnson rule the optimal solution is to process the jobs according to decreasing-b order, resulting in makespan $2^{n+2}-2$.

If the second machine is hurt, then for all jobs $a_j<b_j$ and by Johnson rule the optimal solution is to process the jobs according to increasing-a order, resulting in makespan $2^{n+2}-2$.

Scheduling 4:

Let C* be the value of an optimal solution for the problem $P|pmtn|C_{max}$ with m>1 machines and n>m jobs. Assume that all job lengths are increased by one, that is, for every job, $p'_j=p_j+1$.
Denote by C'* the value of an optimal solution for $P|pmtn|C_{max}$ on the resulting instance.

What is the minimal value of C'*-C*?

Scheduling 4:

It is known that the makespan equals $\max(\max_j p_j , 1/m \sum_j p_j)$. Due to the change, $\sum_j p_j$ is increased by n and since n>m, $1/m \sum_j p_j$ is increased by more than 1. On the other hand, $\max_j p_j$ is increased by 1. Therefore, the new makespan is increased by at least 1.

This is tight, for example if m=2, n=3 and $p_1=100$, $p_2=p_3=1$.

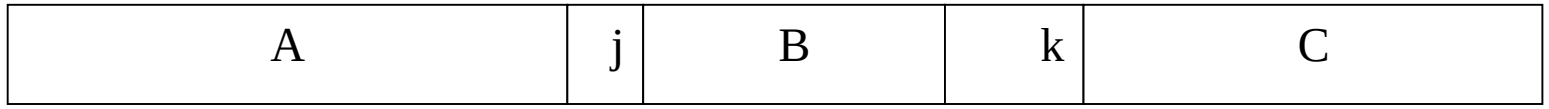
Scheduling 5:

Consider the problem $1||\Sigma w_j T_j$ in which jobs have processing times p_j, due-dates d_j, and weight w_j. The goal is to find a schedule of the jobs on a single machine, in a way that minimizes the total weighted tardiness ($T_j=\max(0, Cj-dj)$).

Prove: If for two jobs j,k it holds that $d_j \leq d_k$, $p_j \leq p_k$ and $w_j \geq w_k$ then there exists an optimal schedule in which j is processed before k.

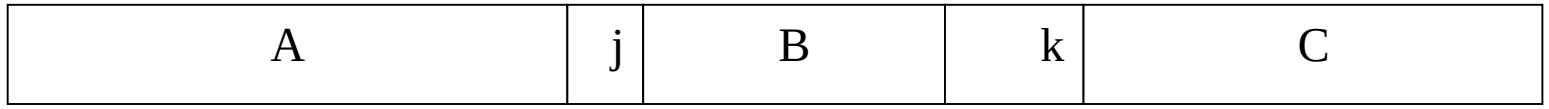
Scheduling 5 (solution):

The proof uses exchange argument. Let S be a schedule in which $d_j \leq d_k$, $p_j \leq p_k$, $w_j \geq w_k$,and k is processed before j (they are not necessarily adjacent). Consider the schedule S' in which the two jobs swap locations. We show that S' is not worse than S.
The completion times of the jobs that were processed before k (the set A) or after j (the set C) do not change. The completion times of the jobs that were processed between k and j (the set B) is shortened by $\Delta p = p_k - p_j$. Therefore we only need to show that the total penalty due to j and k is not increased.



Scheduling 5 (solution, cont'):

If both jobs were not late in S then both are not late in S' ($d_j \le d_k$). Note that $C'_j = C_k - \Delta p$ and $C'_k = C_j$. If both were late in S and are late in S', then their shared penalty is reduced by $(p_B+p_k)w_j-(p_B+p_j)w_k$, which is non-negative. If both were late in and only one of them is late in S' then the reduction is just increased. If only j was late in S then we save $(p_B+p_k)w_j$ and add at most $(p_B+p_j)w_k$ to the total penalty (k might be late or not in S'). Again, the total reduction is non-negative. Finally, note that it is not possible that only k was late in S (since $d_j \le d_k$) .



Facility Location 1:

The following is the 2- approximation algorithm for k-center studied in class.
1. Choose the first center arbitrarily.
2. At every step, choose the vertex that is furthest from the current centers to become a center.
3. Continue until k centers are chosen.

For any given n_0, describe a graph with $n \geq n_0$ vertices, such that **every** run of the algorithm with k=1 and k=2 is optimal, and every run with k=3 provides a 1.5-approximation.
Describe the graph, an optimal solution, the algorithms' solution, and conclude the approximation ratio.

Facility Location 1, solution:

Given n0, let n be a multiple of 12 at least n0. Consider a cycle of n nodes. Any node is an optimal 1-center (max distance =n/2).

Any two nodes that are n/2 apart from is other form an optimal 2-center. (max distance = n/4).

An optimal 3-center consists of three nodes that are n/3 apart (max-distance =n/6) while the third center of the algorithm does not affect the max distance (which remains n/4).

Facility Location 2:

Let G be a directed graph with positive edge weights.
The solution for the absolute-1-center has value z.
The solution for the vertex-1-center has value 2z.
Prove that G consists of a single edge.

Facility Location 2, solution:

Direction 1: If G is just an edge (u,v) of length 2z then the absolute-center is in the middle of the edge (and m(x)=z) while the optimal vertex-center is either u or v for which m(v)=m(u)=2z

Direction 2: Given the above, assume towards contradiction that G consists of more than a single edge. Let v a vertex center. m(v)=2z so there is some vertex u for which d(u,v)=2z.

Facility Location 2, solution:

Let x be an absolute center. d(x,u) as well as d(x,v) are at most z so the path from u to v that goes through x has length at most 2z. Also, since d(u,v)=2z , the path from u to v that goes through x must have length exactly 2z. Assume x is the local center of an edge (p,q). Since the graph includes more than one edge (p,q) is not the edge (v,u) and therefore c(p,q)<2z – because the edge is just a part of the path from u to v (it might be that p=u of q=v but not both).
This contradicts the known upper bound on the local center of an edge:

$$m(x) \geq \frac{m(p)+m(q)-c(p,q)}{2}$$

$m(q) \geq 2z$, $m(p) \geq 2z$, $c(p,q)<2z$ so m(x) must be strictly larger than z.

Facility Location 3:

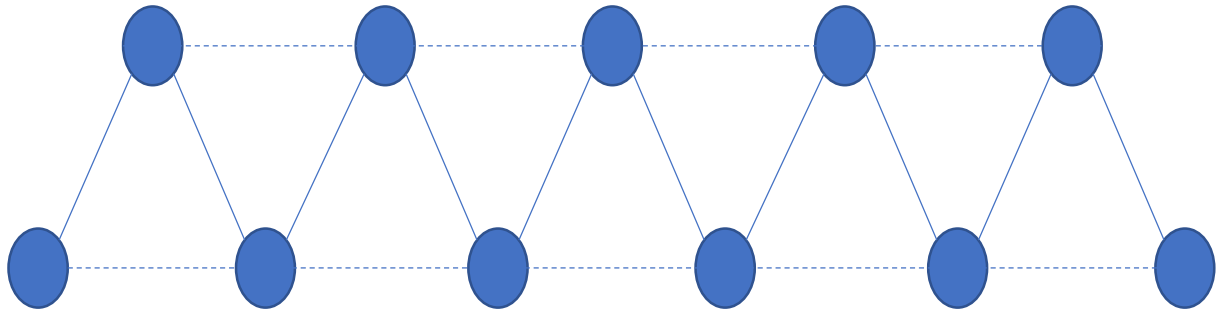
Consider a bipartite G=(V,E) with $V=V_1 \cup V_2$, $E \subseteq V_1 \times V_2$. It is known that all nodes have the same weight and all edges have the same length.
Prove or give a counter example: In any optimal solution for the 2-median problem on G, one median is placed on V_1 and one median is placed on V_2 .

False: take a path v1---v2----v3----v4----v5. This is a bipartite with V1={v1,v3,v5}, V2={v2,v4}. An optimal solution is picking v2 and v4 as the medians.

An even smaller counter example: v1—v2—v3. The set {v1,v3} forms an optimal solution (any set of two vertices is optimal).

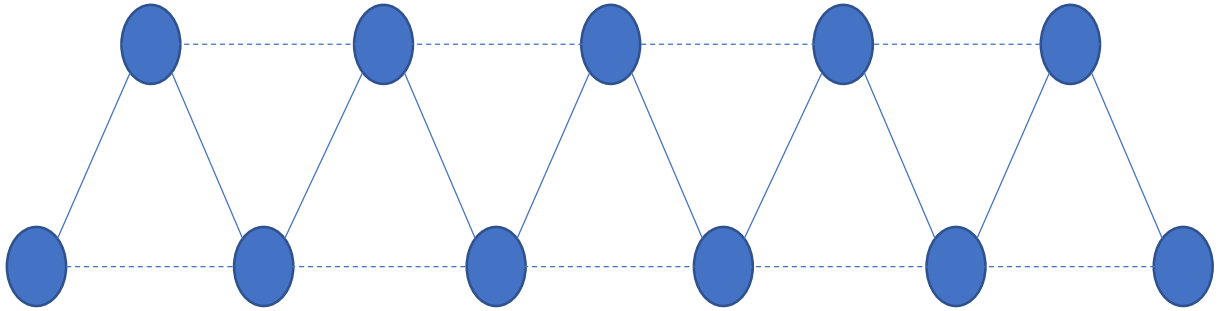
Facility Location 4:

The graph in the figure has 2n+1 nodes, all solid edges have length 1, all dashed edges have weight 1+ε. The lengths of missing edges is the length of the shortest path connecting their endpoints (to obey the triangle ineq.)



Show (using this graph) that the analysis of the 3/2-approximation algorithm for TSP is tight.

Facility Location 4 (solution):



The only MST consists of all solid lines, it has length 2n. No shortcuts are needed. The first and last nodes have odd degree. The edge connecting them has length $(1+\varepsilon)(n+1)$. All together, the TSP path detected by the algorithm has length ~3n.

An optimal tour consists of the dashed edges + the leftmost and rightmost edges. Its length is $(2n-1)(1+\varepsilon)+2 \sim 2n$

Algorithmic Game Theory 1:

In class we proved that LPT produces a NE in job scheduling games on identical machines. In this question you will extend this result for machines with different speeds. Let s_i denote the speed of machine i. It means that it takes p_j/s_i time units to process job j on M_i. The completion time of M_i is $Ci=\sum_{j \text{ assigned to } i} p_j/s_i$. In our game, this is also the cost of each of the jobs assigned to M_i.

Algorithm LPT sorts the jobs such that $p_1 \geq p_2 \geq \ldots \geq p_n$. The jobs are assigned greedily on the machines. Jj is assigned to the machine that minimizes its cost at the assignment time (ties are broken in favor of a fast machine).

Algorithmic Game Theory 1:

1. Let m=2, s_1=1, s_2=2. Show the LPT-schedule of an instance with 8 jobs having lengths {12, 8, 8, 5, 3, 3, 2, 2}.

2. Prove that for any m and n, LPT produces a Nash-equilibrium. That, is, after the assignment is over, no single job would like to migrate to another machine.

Directions: Use induction on the number of jobs already assigned. Formally, show that for each k=1,2,..,n, after J_k is assigned, no job in {1,..,k} can improve its cost by migrating.

Algorithmic Game Theory 1:

LPT schedule:
The slow machines (M1) is assigned jobs of lengths {8,3,2}.
The fast machine (M2) is assigned jobs of lengths {12,8,5,3,2}.
The makespan is achieved on M2 (=30/2=15}.

Algorithmic Game Theory 1:

b. **Base case**: After the first job is assigned on the fastest machine it would not benefit from moving to a slower machine.

Step: Assume that after J_{k-1} is assigned, no job in {1,..,k-1} can improve its cost by migrating. Consider the assignment of J_k. Assume it is assigned to machine M. Clearly, J_k itself won't like to migrate as it is assigned by the algorithm on the machine minimizing its cost. Also any job out of M won't like to migrate because no one wanted to migrate before J_k was added, and the load on M just increased. Therefore, the only candidates for migration are the jobs on M. Assume that J_z (z<k) prefers to migrate from M to M'. Let s,s' be the speeds of M,M'. Let C,C' denote the load on M,M' before J_k is assigned. J_k is assigned to M because $C+p_k/s \leq C'+p_k/s'$. J_z wants to migrate therefore $C+p_k/s > C'+ p_z/s'$. Combining the above inequalities we get $p_k > p_z$ contradicting the LPT order.

Algorithmic Game Theory 2:

Considers the following scheduling game on related machines:
The input is given as an n×m matrix that includes the processing times p_{ij}.
For a given assignment, the cost of a job is the total processing time of the jobs assigned to its machine.

In this problem we are going to prove that the price of anarchy is unbounded.
Given r, complete the processing times in the following matrix, such that the price of anarchy in the resulting game (with two jobs and two machines) is r.
In other words – for the instance you specify, there exists a NE assignment in which the maximal cost of a jobs is r while in the optimal assignment, the maximal cost of a job is 1.

Complete the table, and describe the two assignments.

pij	M1	M2
J1		
J2		

Algorithmic Game Theory 2:

For every job, there exists one machine on which it is fast and one machine on which it is slow. When both jobs are assigned on their slow machine the makespan is r and this is a NE. A better NE, with makespan=1, is when both jobs are assigned on their fast machine.

pij	M1	M2
J1	1	r
J2	r	1

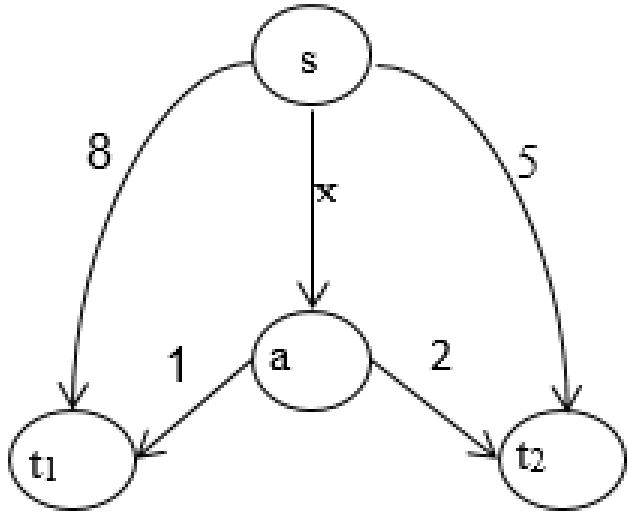
Algorithmic Game Theory 3:

The routing problem with **profit sharing** is defined similarly to the routing problem with cost sharing: the weights on the edges are profits the users share. That is, if an edge of weigh p_e is used by $x_e > 0$ players, then each of them earn p_e / x_e. Each players must select a simple (si,ti)-path. The goal of each player is, of course, to find a path that maximizes her profit.

A profile is a social optimum if the total profit of all players is the maximum possible.

A profile is a Nash Equilibrium if no player can switch to a path with higher revenue.

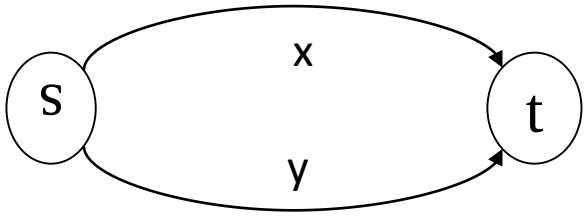
Algorithmic Game Theory 3:



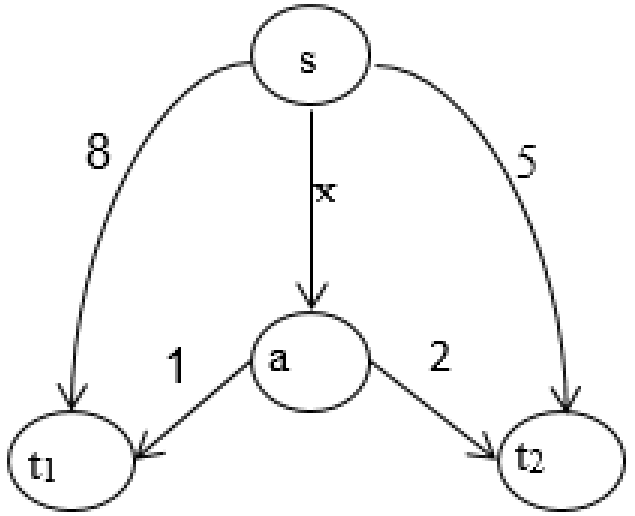
Consider the following profit-sharing game.
1. What is the social optimum?
Distinguish between different values of x.

2. For which values of x, the profile in which both players are using the edge (sa) is a NE?

3. For the following symmetric game and k=2. What is the maximal possible ratio between the social optimum and the total players' profit in a NE? Your solution may be a constant or a function of x,y.



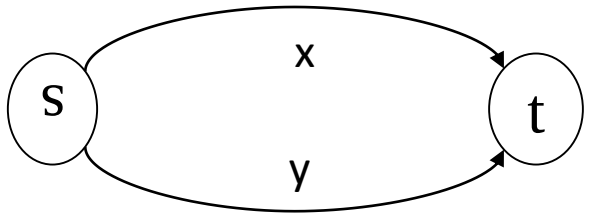
Algorithmic Game Theory 3:



a. If both players use the inner path then the total profit is x+3. If only the first player uses the inner path, then the total profit is x+6. If only the second player uses the inner path, then the total profit is x+10. If both use the outer paths then the total profit is 13. Therefore the social optimum is max(13,x+10). In other words: if $x \leq 3$ then the social opt is 13, and otherwise it is x+10.

b.The first agent will not switch as long as $x/2+1 \geq 8$, that is $x \geq 14$.
The second agent will not switch as long as $x/2+2 \geq 5$, that is $x \geq 6$.
Combining both conditions, routing through the middle path is a NE for any $x \geq 14$.

Algorithmic Game Theory 3:



The social optimum is always x+y.
Both players routing via y is a NE only if $x \leq y/2$.
The ratio is $(x+y)/y \leq 1.5y/y=1.5$.

Similarly, Both players routing via x is a NE only if $y \leq x/2$. The ratio is at most $(x+y)/x \leq 1.5$

If one player routes via x and one via y, then the ratio is 1.

The maximal ratio is max(1.5, 1.5, 1)=1.5

Stable matching 1:

Consider an instance in which all the boys have the same ranking of the girls (say, g1,g2,…,gn.), and all the girls have the same ranking of the boys (say b1,b2,…,bn).

How many days it takes for TMA to terminate on this instance? Describe how the algorithm proceeds and what are the created couples.

Solution: In the first day, all the boys go to g1. On day 2, b1 will return to g1 and n-1 boys will go to g2. In general, on day k, boys 1..k-1 are matched with girls 1..k-1 and all n-k+1 boys go to girl k. After n days, for every j, bj is matched with gj.

Stable matching 2:

Let A and B be two different stable pairings of n boys with n girls. Consider the following way to build a new pairing C from A and B. For each boy, A pairs him with some girl g_A and B pairs him with some girl g_B; to construct C we give him his favorite of the two girls g_A and g_B. (g_A and g_B might be the same girl, which is fine.) It is not even obvious that C is a pairing-perhaps some girl will get matched with two boys. Strangely, C is not only a pairing, it is stable! Prove that it is a pairing and that it is stable.

Stable matching 2 (solution):

Clearly, each boy is matched in C with a single girl. Assume that C is not a pairing, that is, some girl, say g1, is matched with both b1 and b2 in C.
both b1 and b2 prefer g1 over their mate in the additional pairing.
If g1 prefers b1 over b2 then the pairing A or B in which she is matched with b2 is not stable (b1-g1 form a rouge couple).
Otherwise g1 prefers b2 over b1 and then the pairing A or B in which she is matched with b1 is not stable (b2-g1 form a rouge couple).

Stable matching 2 (solution):

Assume C is not stable, and let (b1-g1) be a rouge couple.
More specifically, assume b1 is matched with g2 but prefers b1 and g1 is matched with b2 but prefers b1.
Assume without loss of generality that b1-g2 form a pair in A. If b2-g1 is a pair in A then A is not stable. So b2-g1 is a pair in B. Assume b1 is matched with g3 in B. We know he prefers g1 (selected her for C) , and prefers g1 even more (they form a rouge couple in C). So b1's list includes g1,g2,g3 in this order while g1's list includes b1,b2 in this order. This implies that B is not stable. A contradiction.

Packing 1:

Present an instance for Bin-Packing for which First-fit decreasing uses 3 bins, while First-Fit uses only two bins.
Describe the sequence, the packing of FFD and the packing of FF.

Solution:

0.5, 0.3, 0.4, 0.2, 0.2, 0.2, 0.2

Packing 2:

Assume that the FPTAS for knapsack is performed on an instance with many items having size 10 and profit 6, and many items having size 16 and profit 15.
The knapsack has capacity 160.
What would be a possible bad choice of k?

Solution:

With k=5 the FPTAS will consider items of size 10 and profit 10, and items of size 16 and profit 15. With these rounded values, items of the first type are more profitable and the FPTAS will pack 16 item with profit 6 (rounded to 10) and will achieve Profit=96. An optimal solution packs 10 items with profit 15. OPT=150.