

Post-Homework Practice Problem Solutions

WARNING: if you haven't at least tried hard to solve the practice problems before reading these solutions, you are missing the point. If you can't make *any* progress, talk to me or to the TAs before reading these solutions. Otherwise, you should come up with a solution of your own that you can compare to the one shown here.

1. Let X be an instance of PARTITION. We construct an instance (S, m) of the bin-packing optimization problem as follows. For each $x_i \in X$, add an item i to S with size $\ell_i = x_i$. Finally, set $m = |X|/2$.

Claim: (S, m) has a solution that uses 2 bins iff X can be partitioned.

Pf: in one direction, if $X = Y + Z$, where $|Y| = |Z| = |X|/2$, then put the items corresponding to Y in bin 1, and put the items corresponding to Z in bin 2. Conversely, if (S, m) has a solution with 2 bins, then each bin must be completely full. Construct Y and Z from the items in bins 1 and 2, respectively; then Y and Z form a partition of X .

Let A be a poly-time approximation for bin-packing with ratio $3/2 - \epsilon$. If we run A on (S, m) , it is guaranteed to return a solution that uses at most $(3/2 - \epsilon)2 = 3 - 2\epsilon$ bins. Since every feasible solution uses an integer number of bins, the algorithm must return the optimal packing.

Conclude that, under the above reduction, A can be used to solve the NP-complete problem PARTITION, and so $P = NP$.

2. The result of the previous problem postulates a bin-packing algorithm A that can get a good approximation for *every* problem instance. In particular, for any instance whose optimal solution uses $B \geq 2$ bins, A could get within the specified ratio. (We used this fact for $B = 2$ in our proof that A cannot exist unless $P = NP$.)

In contrast, the FFD algorithm only guarantees a ratio of $11/9B + 4$. For $B = 2$, FFD is only guaranteed to give a ratio of $11/9 \cdot 2 + 4$, and so to produce a solution with at most 6 bins. In fact, this ratio guarantee does not get below $3/2$ for any problem with $B < 15$.

We say that FFD guarantees an *asymptotic approximation ratio* of $11/9$ for bin-packing, since this ratio is only achieved in the limit of very large problem instances.

Note: subsequent work on the FFD algorithm has reduced the additive constant to 3. A result from 1991 reduces the constant to 1, though Johnson's 1997 survey of bin-packing algorithms suggests he doesn't believe that result. If correct, it would achieve a ratio of better than $3/2$ for any $B \geq 4$.

3. Consider the following LP-rounding algorithm for P . First, compute the LP optimum \bar{X} . Then, form an integer solution \hat{X} as follows: for every i , if $\bar{x}_i > 0$, set $\hat{x}_i = 1$. Return the solution \hat{X} .

We first note that \hat{X} is feasible. Indeed, every constraint in the problem is a *lower bound*, so if the constraint is satisfied in the (feasible) LP optimum \bar{X} , then increasing the values of the variables cannot make it unsatisfied.

To show the claimed ratio, let $S(X) = \sum_i c_i \cdot x_i$ be the value of any solution to P . Let X^* be an optimal integer-valued solution to P . We know that $S(\bar{X}) \leq S(X^*)$, since the LP optimum is no worse than the IP optimum. To obtain an upper bound, note that, because no non-zero \bar{x}_i is less than $1/k$, our rounding procedure increases the value of all such \bar{x}_i by a factor of at most k . Conclude that

$$\begin{aligned} S(\hat{X}) &= \sum_i c_i \cdot \hat{x}_i \\ &= \sum_{\bar{x}_i > 0} c_i \cdot \hat{x}_i \\ &\leq \sum_{\bar{x}_i > 0} c_i \cdot k \cdot \bar{x}_i \\ &= k \sum_i c_i \cdot \bar{x}_i \\ &= kS(\bar{X}) \\ &\leq kS(X^*). \end{aligned}$$

4. (a) The generalization of the IP for m machines includes the following constraints:

- For each job i ,

$$\sum_j x_{ij} = 1.$$

- For each machine j ,

$$\sum_i p_{ij} x_{ij} \leq t.$$

There are now $nm + 1$ variables (t plus indicators for each job, for each machine), $n + m$ “nontrivial” constraints, and $nm + 1$ constraints of the form $v \geq 0$. Hence, there exists an optimal solution to the LP relaxation of this IP that makes at least

$$nm + 1 - (n + m)$$

variables zero. Now t cannot be zero, so conclude that at most $n + m - 1$ of the x_{ij} s can be nonzero.

Now, how many jobs can be split across machines? Let a and b be the number of split and unsplit jobs, respectively. We have that $a + b = n$, since there are n total jobs. Each split job makes at least 2 x_{ij} s nonzero, while each unsplit job makes exactly one variable nonzero, so we have that $2a + b \leq n + m - 1$. Subtracting the equation from the inequality gives us $a \leq m - 1$; that is, at most $m - 1$ jobs can be split.

(b) We generalize our two-machine algorithm as follows. First, set up the general IP with constraints as described above and objective t . Second, solve the LP relaxation to obtain an LP solution \bar{X} . Third, we construct a fully integral solution as follows:

- Let S be the set of at most $m - 1$ jobs split by the LP solution, and let N be the remaining jobs.

- Let X_N be the (integral) schedule given by \overline{X} to the jobs in N .
- Consider *all possible* ways of scheduling the jobs of S on m , and let X_S be the optimal solution found.
- Finally, place the jobs of N and S on the machines specified by X_S and X_N , respectively, and return the joint schedule \hat{X} .

We claim that this algorithm is a 2-approximation. Let $S(X)$ be the max length of schedule X , and let X^* be an optimal schedule. Because the schedule X_N is a subset of \overline{X} , we have

$$S(X_N) \leq S(\overline{X}) \leq S(X^*).$$

Moreover, because S is a subset of the input set of jobs, we have

$$S(X_S) \leq S(X^*).$$

Conclude that

$$\begin{aligned} S(\hat{X}) &\leq S(X_S) + S(X_N) \\ &\leq S(X^*) + S(X^*) \\ &= 2S(X^*). \end{aligned}$$

The cost of the algorithm includes $O(mn)$ to set up the IP, the cost of solving the LP, the $O(nm)$ cost of identifying the sets N and S , and the cost of trying all possible schedules for S . To bound the number of such schedules, note that the number of ways to assign $m - 1$ jobs to m machines is m^{m-1} , and that we can surely determine the cost of each such schedule in time $O(m)$. Conclude that the cost of the algorithm is $O(mn + m^m)$, which is fine because m is a constant!

Note: there is actually a 2-approximation algorithm for any number of machines that is polynomial in both n and m . The algorithm relies on a faster way to find a good assignment of the $m - 1$ split jobs to processors. For details, see Chapter 17 of Vazirani's book, or Chapter 1 of Hochbaum's book.