Introduction to Algorithms **Solution Sketches for Problem Set 11**
CS 4820, Spring 2018

(1) *(10 points)*
For each of the following optimization problems, present an integer program whose optimum value matches the optimum value of the given problem. The combined number of variables and constraints in your integer program should be polynomial in the size of the given instance of the optimization problem.

(i). SET COVER. Given a universal set $\mathcal{U}$ and a collection of subsets $S_1, S_2, \ldots, S_m \subseteq \mathcal{U}$, what is the minimum size of a subcollection $\{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ whose union is $\mathcal{U}$?

ANSWER:

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} x_i \\ \text{s.t.} \quad & \sum_{i\,:\,u \in S_i} x_i \geq 1 \ \text{ for all elements } u \in \mathcal{U} \\ & x_i \in \{0,1\} \quad \text{for } i = 1, 2, \ldots, m \end{aligned}$$

INTERPRETATION: Decision variable x_i equals 1 if set S_i is included in the set cover, 0 otherwise.

(ii). INDEPENDENT SET. Given a graph $G = (V, E)$, find an independent set of maximum cardinality.

ANSWER:

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & x_u + x_v \leq 1 \ \text{ for all edges } e = (u, v) \\ & x_v \in \{0,1\} \quad \text{for all vertices } v \in V \end{aligned}$$

INTERPRETATION: Decision variable x_v equals 1 if v is in the independent set, 0 otherwise.

(iii). MAX-3SAT. Given a set of Boolean variables $x_1, x_2 \ldots, x_n$ and a set of clauses $C_1, C_2, \ldots, C_m$ each consisting of a disjunction of 3 literals from the set $\{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$, what is the maximum number of clauses that can be satisfied by a truth assignment?

ANSWER:

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} z_j \\ \text{s.t.} \quad & z_j \leq y_a + (1 - y_b) + y_c \ \text{ if, for example, clause } C_j = (x_a \vee \bar{x}_b \vee x_c) \\ & (\ldots \text{one constraint like the one above for each clause}) \\ & y_i \in \{0,1\} \ \text{ for all variables } x_i \\ & z_j \in \{0,1\} \ \text{ for all clauses } C_j \end{aligned}$$

INTERPRETATION: Decision variable y_i is 1 if x_i is TRUE in the truth assignment, 0 otherwise. Decision variable z_j is 1 if clause C_j is satisfied by this truth assignment, 0 otherwise. The constraints ensure that the objective function "gives us credit" only for the clauses which are

satisfied by the truth assignment represented by the y_i decision variables.
REMARK: The difficulty in this problem is expressing the objective as a *linear* function of the decision variables, while also expressing the constraints as *linear* constraints. For example, if we had written

$$z_j = \min\{1, y_a + (1 - y_b) + y_c\} \tag{1}$$

instead of the pair of constraints

$$\begin{aligned} z_j &\leq y_a + (1 - y_b) + y_c \\ z_j &\in \{0, 1\} \end{aligned}$$

then the constraint (1) would not be a linear constraint, so the solution would not be a valid integer program.

(iv). MAX-CUT. Given an undirected graph $G = (V, E)$, what is the maximum size of a cut? (A cut (A, B) is any partition of the vertex set V into two nonempty subsets. The size of a cut is equal to the number of edges with one endpoint on each side of the partition.)

ANSWER:

$$\begin{aligned} \max \quad & \sum_{e \in E} y_e \\ \text{s.t.} \quad & y_e \leq x_u + x_v \text{ for all edges } e = (u, v) \\ & y_e \leq 2 - x_u - x_v \text{ for all edges } e = (u, v) \\ & x_v \in \{0, 1\} \text{ for all vertices } v \in V \\ & y_e \in \{0, 1\} \text{ for all edges } e \in E \end{aligned}$$

INTERPRETATION: Decision variable x_v is 1 if $x \in A$ and 0 if $x \in B$. Decision variable y_e is 1 if one endpoint of e is in A and the other is in B, otherwise $y_e = 0$. The constraint $y_e \leq x_u + x_v$ guarantees that $y_e = 0$ when both endpoints of e lie in B, and $y_e \leq 2 - x_u - x_v$ guarantees that $y_e = 0$ when both endpoints of e lie in A.
REMARK: As before, the difficulty in this problem lies in expressing the constraint "$y_e = 1$ only if edge e crosses the cut" using *linear* inequalities. The equations $y_e = x_u \oplus x_v$, $y_e = (x_u - x_v)^2$, and $y_e = |x_u - x_v|$ all express this constraint when $x_u, x_v \in \{0, 1\}$, but none of these equations is a linear equation or inequality.

(2) *(15 points)*
Recall that in the Knapsack Problem, one is given a set of items numbered $1, 2, \ldots, n$*, such that the* i^{th} *item has value* $v_i \geq 0$ *and size* $s_i \geq 0$*. Given a total size constraint* B*, the problem is to choose a subset* $S \subseteq \{1, 2, \ldots, n\}$ *so as to maximize the combined value,* $\sum_{i \in S} v_i$*, subject to the size constraint* $\sum_{i \in S} s_i \leq B$*. The input is assumed to satisfy* $s_i \leq B$ *for all* $i \in \{1, \ldots, n\}$*.*

(a) *Consider the following* greedy algorithm*, GA.*

(i). For each i*, compute the* value density $\rho_i = v_i/s_i$*.*

(ii). Sort the remaining items in order of decreasing ρ_i*.*

(iii). Choose the longest initial segment of this sorted list that does not violate the size constraint.

Also consider the following even more greedy algorithm*, EMGA.*

(i). Sort the items in order of decreasing v_i.

(ii). Choose the longest initial segment of this sorted list that does not violate the size constraint.

For each of these two algorithms, give a counterexample to demonstrate that its approximation ratio is not bounded above by any constant C. (Use different counterexamples for the two algorithms.)

(b) *Now consider the following algorithm: run GA and EMGA, look at the two solutions they produce, and pick the one with higher total value. Prove that this is a 2-approximation algorithm for the Knapsack Problem, i.e. it selects a set whose value is at least half of the value of the optimal set.*

(c) *By combining part (b) with the dynamic programming algorithm for Knapsack presented in class, show that for every $\delta > 0$, there is a Knapsack algorithm with running time $O(n^2/\delta)$ whose approximation ratio is at most $1 + \delta$. [Recall that the algorithm presented in class had running time $O(n^3/\delta)$.] In your solution, it is not necessary to repeat the proof of correctness of the dynamic programming algorithm presented in class, i.e. you can assume the correctness of the pseudopolynomial algorithm that computes an exact solution to the knapsack problem in time $O(nV) = O(n\sum_{i=1}^{n} v_i)$, when the values v_i are integers.*

Solution.

(a) A counterexample for GA is given by an instance with two elements: ($v_1 = 2$, $s_1 = 1$) and ($v_2 = 2C + 1, s_2 = 2C + 1$). If the knapsack capacity is $2C + 1$, then GA will pick the first element and then the second one won't fit. The value obtained will be only 2, whereas value $2C + 1$ would be obtained by taking the second element instead.

A counterexample for EMGA is given by an instance with $2C + 2$ elements: one element with ($v_1 = 2, s_1 = 2C + 1$) and $2C + 1$ elements each with ($v_i = 1, s_i = 1$). If the knapsack capacity is $2C + 1$, then EMGA will pick the first element and then all the remaining ones won't fit. The value obtained will be only 2, whereas value $2C + 1$ would be obtained by taking all elements except the first one.

(b) We'll prove that

$$\mathrm{GA} + \mathrm{EMGA} \geq \mathrm{OPT} \tag{2}$$

from which is follows that $\max\{\mathrm{GA}, \mathrm{EMGA}\} \geq \frac{1}{2}\mathrm{OPT}$. To prove (2), let $i_1, \ldots, i_k$ denote the sequence of items selected by GA (in the order selected) and let i_{k+1} be the next element that GA would have selected, if it hadn't run out of space. The EMGA algorithm gets a value at least as high as the value of item i_{k+1}, and the value of items $i_1, \ldots, i_k$ is, of course, exactly the value achieved by the GA algorithm. On the other hand, the combined value of items $i_1, \ldots, i_k, i_{k+1}$ is greater than that of the optimal knapsack solution, since OPT has a strictly smaller combined size, and any elements of OPT that are not among $\{i_1, \ldots, i_{k+1}\}$ have a value density that is strictly smaller than the least dense element of that set. In more detail, let

$$\begin{aligned} A &= \mathrm{OPT} \cap \{i_1, \ldots, i_{k+1}\} \\ B &= \mathrm{OPT} \setminus A \\ C &= \{i_1, \ldots, i_{k+1}\} \setminus A \\ \rho &= \rho_{i_{k+1}} \end{aligned}$$

We have

$$\begin{aligned} \sum_{i \in A} s_i \; + \; \sum_{i \in B} s_i \leq B \leq \sum_{i \in A} s_i \; + \; \sum_{i \in C} s_i \\ \sum_{i \in B} s_i \leq \sum_{i \in C} s_i \\ \sum_{i \in B} v_i \leq \rho \sum_{i \in B} s_i \leq \rho \sum_{i \in C} s_i \leq \sum_{i \in C} v_i \end{aligned}$$

since $v_i \leq \rho s_i$ for all $i \in B$ while $v_i \geq \rho s_i$ for all $i \in C$. Hence

$$\text{OPT} = \sum_{i \in \text{OPT}} v_i = \sum_{i \in A} v_i + \sum_{i \in B} v_i \leq \sum_{i \in A} v_i + \sum_{i \in C} v_i = \sum_{j=1}^{k+1} v_{i_j} \leq \text{GA} + \text{EMGA}$$

which establishes (2).

(c) In class on April 27, an algorithm was presented for the special case of Knapsack in which all values are integers. The algorithm is a dynamic program, and its running time is $O(nV)$ where V is the sum of the values. Consider the following modification of that algorithm, which assumes that we are given a knapsack instance with budget B, and with n items having integer values v_i and sizes s_i, and in addition we are given an "upper bound parameter" — a positive integer denoted by Q, representing an upper bound on the value of the optimum knapsack solution.

---

for $i = 0, \ldots, n$ **do**
  for $j = 0, \ldots, Q$ **do**
    if $j = 0$ **then**
      $U[i, j] = 0$
      $S[i, j] = \emptyset$
    else if $i = 0$ **then**
      $U[i, j] = \infty$
      $S[i, j] = \emptyset$
    else
      $j' = \max\{0, j - v_i\}$.
      $U[i, j] = \min\{U[i-1, j], s_i + U[i-1, j']\}$.
      $$S[i, j] = \begin{cases} S[i-1, j] & \text{if } U[i, j] = U[i-1, j] \\ \{i\} \cup S[i-1, j'] & \text{if } U[i, j] < U[i-1, j] \end{cases}$$
    end if
  end for
end for
Let $j^* = \max\{j \mid U[n, j] \leq B\}$.
Output $S[n, j^*]$.

This dynamic program runs in time $O(nQ)$ and satisfies the following guarantee: the output is always a feasible solution to the knapsack problem, and for knapsack instances whose optimum value is less than or equal to Q, the output is an optimal solution.

The running time guarantee follows by counting loop iterations and assessing the running time per loop iteration. To justify the running time bound of $O(1)$ per loop iteration, one needs to encode the sets $S[i, j]$ in a way that allows constructing the union of a set with a singleton set in constant time. To do so, we can represent a set as a linked list of items, and add a singleton to a set by prepending the singleton at the head of the linked list, without copying all the other items in the list.

The correctness guarantee of the dynamic program follows from the assertion: *for all i and j, the table entry $U[i, j]$ stores the minimum combined size of a subset of items $1, \ldots, i$ whose combined value is at least j, and the table entry $S[i, j]$ stores a set that attains this minimum, unless there is no subset of $\{1, \ldots, i\}$ whose combined value is at least j, in which case $U[i, j] = \infty$.* The proof of the assertion is an easy induction on pairs (i, j), in the order that the nested loops process them, and is omitted from this solution.

To design the knapsack approximation algorithm, we will run the above dynamic program on a modified instance in which the sizes and budget are unchanged and the values are modified to

$$\tilde{v}_i = \lceil \epsilon \cdot v_i \rceil$$

where $\epsilon = \frac{(1+\delta)n}{\delta R}$ and R is the value of the knapsack solution selected by the 2-approximation algorithm in part (b). To set the parameter Q, we will choose $Q = 2\epsilon R + n$. The justification for this choice of Q is that it is guaranteed to be an upper bound on maximum the $\tilde{v}$-value of any knapsack solution. In fact, since the algorithm in part (b) is a 2-approximation algorithm, we know that $2R$ is an upper bound on the v-value of any knapsack solution. When we scale down by ϵ this upper bound becomes $2\epsilon R$, and when we apply the ceiling function to obtain the $\tilde{v}$-value of the knapsack solution, this inflate the upper bound $2\epsilon R$ by an amount less than one per element of the knapsack solution, hence less than n in total. Thus, $Q = 2\epsilon R + n$ is an upper bound on the $\tilde{v}$-value of any knapsack solution, as claimed. Applying the correctness guarantee of the dynamic program, we may conclude that it outputs a knapsack solution of maximum $\tilde{v}$-value. In particular, letting S^* denote the optimum knapsack solution with the original values v_i, and letting S denote the output of our approximation algorithm, we have

$$\sum_{i \in S} \tilde{v}_i \geq \sum_{i \in S^*} \tilde{v}_i.$$

Since

$$\frac{1}{\epsilon}\tilde{v}_i \geq v_i \geq \frac{1}{\epsilon}\left(\tilde{v}_i - 1\right)$$

for all i, we have

$$\begin{aligned}
\sum_{i \in S} v_i &\geq \frac{1}{\epsilon} \sum_{i \in S} (\tilde{v}_i - 1) \\
&\geq \left(\frac{1}{\epsilon} \sum_{i \in S} \tilde{v}_i \right) - \frac{n}{\epsilon} \\
&\geq \left(\frac{1}{\epsilon} \sum_{i \in S^*} \tilde{v}_i \right) - \frac{n}{\epsilon} \\
&\geq \left(\sum_{i \in S^*} v_i \right) - \frac{n}{\epsilon}. \qquad (3)
\end{aligned}$$

Now, for our choice of ϵ we have

$$\frac{n}{\epsilon} = \frac{\delta}{1+\delta} \cdot R \leq \frac{\delta}{1+\delta} \cdot \left(\sum_{i \in S^*} v_i \right),$$

since $R \leq \left(\sum_{i \in S^*} v_i\right)$. (Recall that R is the value of one of the greedy knapsack solutions, and $\sum_{i \in S^*} v_i$ is the value of the optimal solution.) Substituting this upper bound for $\frac{n}{\epsilon}$ into inequality (3) above, we obtain

$$\sum_{i \in S} v_i \geq \left(1 - \frac{\delta}{1+\delta}\right) \left(\sum_{i \in S^*} v_i \right) = \frac{1}{1+\delta} \left(\sum_{i \in S^*} v_i \right),$$

from which it follows that S is a $(1+\delta)$-approximation to the optimum knapsack solution.

Finally, the running time of the algorithm is

$$O(nQ) = O(n(2\epsilon R + n)) = O\left(n \cdot \frac{2(1+\delta)n}{\delta R} \cdot R + n^2 \right) = O\left(\frac{n^2}{\delta} \right),$$

since $1 + \delta = O(1)$.