Combinatorial Optimization

Final examination: solutions

1. In the simplex tableau below, several variables are candidates to become basic in the next basic feasible solution.

x_1	x_2	x_3	s_1	s_2	s_3	s_4	z	RHS
-5	0	0	-2	0	0	-6	1	245
5	1	0	1	0	0	-3	0	40
2	0	0	1/2	1	0	2	0	18
1/2	0	0	$-4/3$	0	1	3	0	12
1	0	1	2/3	0	0	1/2	0	24

Which variable should be made basic in the next basic feasible solution if the goal is

(a) for the new objective value to be 285? Why?

(b) for the next basic feasible solution to be degenerate? Why?

(c) to have $s_3 = 0$ in the next basic feasible solution? Why?

(d) to achieve the greatest increase in the objective value? Why?

▷ **Solution.** First we observe that the three variables that are candidates to become basic in the next basic feasible solution are x_1, s_1, and s_4, because these are the columns that have negative entries in the objective row. In the x_1 column, the minimum test ratio is $40/5 = 8$, which occurs in the first row of the body of the tableau, In the s_1 column, there is a tie for the minimum test ratio: $18/(1/2) = 36$ in the second row of the body of the tableau, and $24/(2/3) = 36$ in the fourth row. In the s_4 column, the minimum test ratio is $12/3 = 4$ in the third row. For convenience, these possible pivot locations are marked in the tableau below.

x_1	x_2	x_3	s_1	s_2	s_3	s_4	z	RHS
-5	0	0	-2	0	0	-6	1	245
$\boxed{5}$	1	0	1	0	0	-3	0	40
2	0	0	$\boxed{1/2}$	1	0	2	0	18
1/2	0	0	$-4/3$	0	1	$\boxed{3}$	0	12
1	0	1	$\boxed{2/3}$	0	0	1/2	0	24

(a) If x_1 is chosen as the pivot column, then to make the entry in the objective row zero, we must add the first row of the body of the (given) tableau to the objective row. This will cause the objective value to increase by 40 to 285, as desired. So x_1 should be made basic.

(b) There is a tie for the minimum test ratio in the s_1 column, so pivoting in this column will result in a zero in the RHS column (below the objective value), which means the corresponding basic feasible solution is degenerate. So s_1 should be made basic.

(c) We can make $s_3 = 0$ by kicking it out of the basis. To do so, we will need to pivot in the third row of the body of the tableau. This is possible only if we pivot in the s_4 column, so s_4 should be made basic.

(d) Pivoting in the x_1 column will increase the objective value by 40, as we saw in part (a). Pivoting in the s_1 column will increase the objective value by 72. Pivoting in the s_4 column will increase the objective value by 24. Therefore s_1 should be made basic.

□

2. Consider the following linear program.

$$\begin{aligned} \text{minimize} \quad & 5y_1 + 3y_2 + 6y_3 \\ \text{subject to} \quad & y_1 \qquad + \ y_3 \geq 60 \\ & y_1 + 2y_2 + \ y_3 \geq 72 \\ & 2y_1 + \ y_2 + 2y_3 \geq 84 \\ & \qquad y_2 - \ y_3 \geq 24 \\ & y_1 \geq 0, \quad y_2 \geq 0, \quad y_3 \geq 0. \end{aligned}$$

The optimal feasible solution to this linear program is $y_1 = 60$, $y_2 = 24$, and $y_3 = 0$. Determine the dual linear program and its optimal solution. Use the optimal dual solution to demonstrate that the claimed optimal solution to the primal is indeed optimal.

▷ **Solution.** The dual of the given linear program is

$$\begin{aligned} \text{maximize} \quad & 60x_1 + 72x_2 + 84x_3 + 24x_4 \\ \text{subject to} \quad & x_1 + \ x_2 + \ 2x_3 \qquad \leq 5 \\ & \qquad 2x_2 + \ x_3 + \ x_4 \leq 3 \\ & x_1 + \ x_2 + \ 2x_3 - \ x_4 \leq 6 \\ & x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0. \end{aligned}$$

By complementary slackness applied to the primal variables and the dual constraints:

1. Either $y_1 = 0$ (which is false) or $x_1 + x_2 + 2x_3 = 5$.
2. Either $y_2 = 0$ (which is false) or $2x_2 + x_3 + x_4 = 3$.
3. Either $y_3 = 0$ (which is true) or $x_1 + x_2 + 2x_3 - x_4 = 6$.

Therefore we can deduce that the optimal dual solution must satisfy the following system of equations.

$$\begin{cases} x_1 + \ x_2 + 2x_3 \qquad = 5 \\ \qquad 2x_2 + \ x_3 + x_4 = 3. \end{cases} \tag{$\star$}$$

By complementary slackness applied to the dual variables and the primal constraints:

1. Either $x_1 = 0$ or $y_1 + y_3 = 60$ (which is true).
2. Either $x_2 = 0$ or $y_1 + 2y_2 + y_3 = 72$ (which is false).
3. Either $x_3 = 0$ or $2y_1 + y_2 + 2y_3 = 84$ (which is false).
4. Either $x_4 = 0$ or $y_2 - y_3 = 24$ (which is true).

So we know that $x_2 = 0$ and $x_3 = 0$. Hence the system ($\star$) from above becomes $x_1 = 5$, $x_4 = 3$, so the optimal dual solution is

$$x_1 = 5, \quad x_2 = 0, \quad x_3 = 0, \quad x_4 = 3.$$

To demonstrate that the claimed optimal primal solution is indeed optimal, we use the optimal dual solution as multipliers for the constraints in the primal:

$$\begin{aligned} 5\,(\ y_1 \qquad + \ y_3 \geq \ 60) \\ 0\,(\ y_1 + 2y_2 + \ y_3 \geq \ 72) \\ 0\,(2y_1 + \ y_2 + 2y_3 \geq \ 84) \\ +\,3\,(\qquad y_2 - \ y_3 \geq \ 24) \\ \hline 5y_1 + 3y_2 + 2y_3 \geq 372. \end{aligned}$$

Because y_1, y_2, and y_3 are all nonnegative, we have

$$5y_1 + 3y_2 + 6y_3 \geq 5y_1 + 3y_2 + 2y_3 \geq 372,$$

so any feasible solution to the primal must have objective value at least 372. It is easy to check that the given primal solution is feasible, and it has objective value $5(60) + 3(24) + 6(0) = 372$, so it is optimal. □

3. Formulate a linear or integer program to answer the following question.

A company manufactures three product lines. Each production run of line i involves a fixed cost F_i and a per-unit cost p_i, so that the cost of x_i units of line i is $F_i + p_i x_i$. These costs, together with the per-unit revenue, are given in the table below.

Product line	Fixed cost	Per-unit cost	Per-unit revenue
1	\$1500	\$45	\$240
2	900	38	190
3	1000	40	210

There are two key production processes, A and B. The time requirements for each product line on each process, and the hours available, are given in the table below.

Process	Product line 1	2	3	Hours available
A	0.25	0.20	0.30	300
B	0.40	0.50	0.20	400

The company will upgrade exactly one of the two processes. The upgrade to Process A will raise the number of effective hours by 20%; that to Process B will raise the number of effective hours by 10%.

Which process should the company upgrade, and how much of each product line should be manufactured in order to maximize the difference between revenue and cost?

▷ **Solution.** For $i \in \{1, 2, 3\}$, let $x_i \geq 0$ denote the number of units of product line i to produce, and let $b_i \in \{0, 1\}$ denote whether to pay the fixed cost for line i. Let $a \in \{0, 1\}$ denote whether to upgrade Process A. (If $a = 0$, then Process B will be upgraded instead.) The objective is to maximize the difference between revenue and cost:

$$\begin{aligned} \text{maximize} \quad & 240x_1 + 190x_2 + 210x_3 && \text{[revenue]} \\ & - 1500b_1 - 900b_2 - 1000b_3 && \text{[fixed cost]} \\ & - 45x_1 - 38x_2 - 40x_3. && \text{[per-unit cost]} \end{aligned}$$

Equivalently,

$$\text{maximize} \quad 195x_1 + 152x_2 + 170x_3 - 1500b_1 - 900b_2 - 1000b_3.$$

We have resource constraints for the available time on Processes A and B. The number of available hours on Process A is $300+30a$, because upgrading Process A gives us 30 additional hours, and the number of available hours on Process B is $400+40(1-a)$, because upgrading Process B (indicated by $a = 0$) gives us 40 additional hours. So the resource constraints for the two processes are

$$\begin{aligned} 0.25x_1 + 0.20x_2 + 0.30x_3 &\leq 300 + 60a, \\ 0.40x_1 + 0.50x_2 + 0.20x_3 &\leq 400 + 40(1 - a). \end{aligned}$$

Furthermore, we cannot produce any units of a product line unless we pay the corresponding fixed cost, so we have

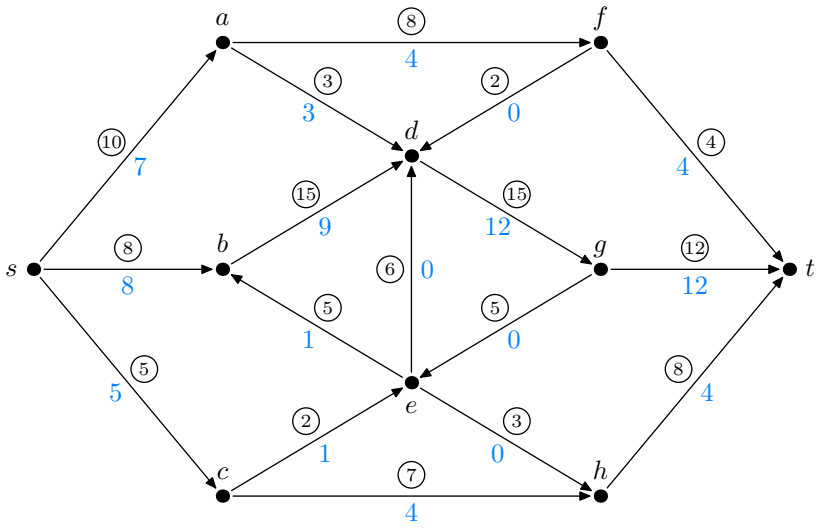
$$\begin{aligned} x_1 &\leq 1{,}000{,}000\, b_1, \\ x_2 &\leq 1{,}000{,}000\, b_2, \\ x_3 &\leq 1{,}000{,}000\, b_3. \end{aligned}$$

These constraints force $x_i = 0$ when $b_i = 0$. The coefficient 1,000,000 here just needs to be large enough so that x_i is not practically constrained when $b_i = 1$.

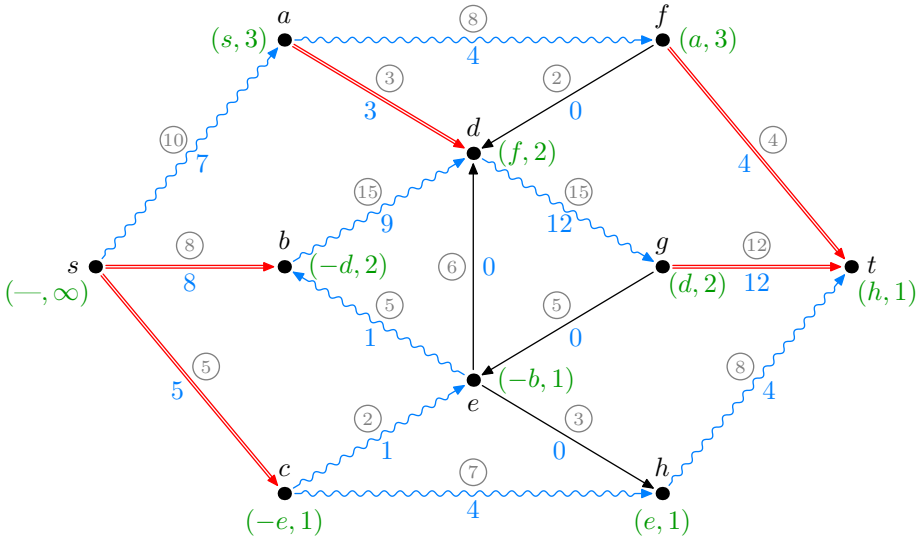
So the complete IP formulation is

$$\begin{aligned}
\text{maximize} \quad & 195x_1 + 152x_2 + 170x_3 - 1500b_1 - 900b_2 - 1000b_3 \\
\text{subject to} \quad & 0.25x_1 + 0.20x_2 + 0.30x_3 \le 300 + 60a \\
& 0.40x_1 + 0.50x_2 + 0.20x_3 \le 400 + 40(1-a) \\
& x_1 \le 1{,}000{,}000\, b_1 \\
& x_2 \le 1{,}000{,}000\, b_2 \\
& x_3 \le 1{,}000{,}000\, b_3 \\
x_1 \ge 0, \quad x_2 \ge 0, \quad x_3 \ge 0, \quad & b_1 \in \{0,1\}, \quad b_2 \in \{0,1\}, \quad b_3 \in \{0,1\}, \quad a \in \{0,1\}. \qquad \square
\end{aligned}$$

4. In the following network, capacities of arcs are shown as circled numbers, and flows along arcs are shown in blue. Is the given s–t flow optimal? If so, prove that it is optimal. If not, find (methodically) a better flow.

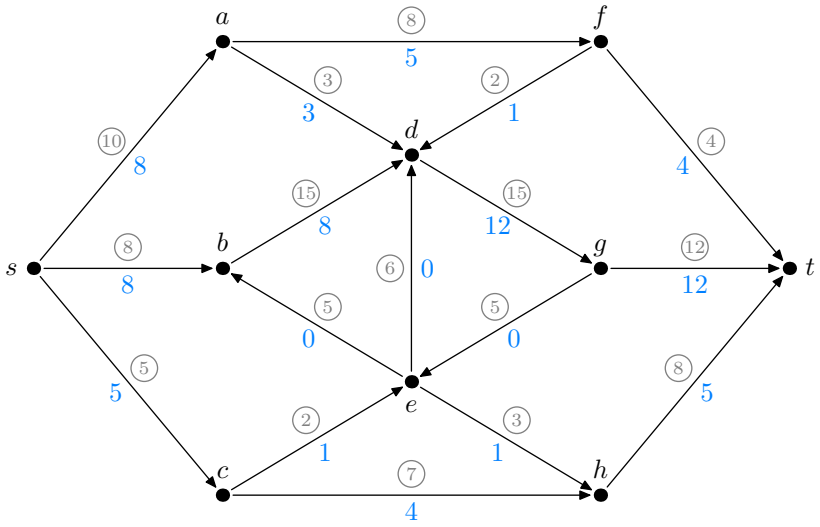


▷ **Solution.** The value of the given s–t flow is 20. The result of one iteration of the Ford–Fulkerson algorithm is shown in the figure at the top of the next page. In this figure, the capacities of the arcs are shown circled in gray, the node labels are in green, and nonzero flows along arcs are in light blue. The node labels are given in the form $(\mathit{from}[i], \mathit{how\text{-}much}[i])$. Arcs with zero flow are drawn as straight black arrows, saturated arcs are drawn as double red arrows, and unsaturated arcs with nonzero flow are drawn as wavy light blue arrows. Below the figure is the *LIST* of vertices built up as the Ford–Fulkerson algorithm progresses through the network. We treat the *LIST* as a queue here: nodes are added to the right-hand side when they are labeled, and they are removed from the left-hand side (indicated by crossing them out) when they are scanned.



LIST: ~~s~~; ~~a~~; ~~f~~; ~~d~~; ~~b~~, ~~g~~; ~~e~~; ~~c~~, ~~h~~; *t*.

The node t received a label, which means that the current flow is not optimal. By following the labels backward from t, we discover the augmenting path s–a–f–d–b–e–h–t. We augment the flow along this path by 1, bearing in mind that the path traverses the arcs (b, d) and (e, b) in the backward direction, so augmenting the flow along the path involves *decreasing* the flow on these arcs. This results in the following flow.



This new s–t flow has value 21, which is better than the flow given in the problem. □

5. Constraint programming is an important area of combinatorial optimization that we did not discuss in this course. An instance of a constraint programming problem (in particular, a constraint satisfaction problem) consists of a finite set X of variables; a finite set $\mathrm{dom}(x)$ for each variable $x \in X$, called the *domain* of x; and a set of *constraints*, each of which specifies, directly or indirectly, a set of allowable combinations of values for a subset of the variables. The objective is to find an assignment of values to all of the variables such that for every variable $x \in X$ the value assigned to x is an element of $\mathrm{dom}(x)$ and such that all constraints are satisfied, or to determine that no such satisfying assignment exists.

One type of constraint is the *alldiff* constraint, which specifies that the values of a subset of the variables must all be different. For example, the constraint *alldiff* (x_1, x_3, x_4, x_8) means that the values assigned to the variables x_1, x_3, x_4, and x_8 must all be different; no two of those variables may be assigned the same value.

Suppose you are given an *alldiff* constraint and the domains of the variables in the constraint. Carefully describe an efficient algorithm to determine whether the *alldiff* constraint is satisfiable (alone, independently of any other constraints that happen to be in the constraint satisfaction problem). Justify that your algorithm is correct. You may use any of the algorithms we discussed in the course as subroutines in your algorithm.

▷ **Solution.** Construct a bipartite graph $G = (U \cup V, E)$ where U is the set of variables in the given *alldiff* constraint and V is the *union* of the domains of these variables. Put an edge between a vertex $x \in U$ (i.e., a variable x) and a vertex $v \in V$ (i.e., a value) if and only if $v \in \mathrm{dom}(x)$. Now find a maximum matching M in the bipartite graph G, using (for example) the augmenting-path algorithm or the network flow formulation we discussed in class. If $|M| = |U|$, then return "yes"; otherwise, return "no."

Why is this algorithm correct? If $|M| = |U|$, then every variable $x \in U$ is matched by M to a value $v_x \in V$. By the construction of G, we must have $v_x \in \mathrm{dom}(x)$ for all $x \in U$. And because M is a matching, no two variables are matched to the same value. So the *alldiff* constraint can be satisfied (in fact, M gives a satisfying assignment of values to the variables). On the other hand, if $|M| < |U|$, then, because M is a maximum matching in G, it is impossible to match each variable $x \in U$ with a value $v_x \in V$ such that $v_x \in \mathrm{dom}(x)$ for all x and the values v_x are all different, so the *alldiff* constraint cannot be satisfied. □

6. In the SET PACKING problem, an instance is a collection $\mathscr{C}$ of finite sets and a positive integer k, and the question is whether $\mathscr{C}$ contains k disjoint sets. Prove that SET PACKING is NP-hard.

▷ **Solution.** To show that SET PACKING is NP-hard, it suffices to give a polynomial-time transformation from a known NP-complete problem to SET PACKING.

Here is a polynomial-time transformation from INDEPENDENT SET to SET PACKING. Let (G, k) be an instance of INDEPENDENT SET, where $G = (V, E)$ is a graph and k is a positive integer. (The question in this instance is whether G contains an independent set of size k.) For $v \in V$, let $E_v = \{\, e \in E : v \in e \,\}$ be the set of edges incident upon v. Construct the instance $(\mathscr{C}, k)$ of SET PACKING, where $\mathscr{C} = \big\{\, \{v\} \cup E_v : v \in V \,\big\}$ and k is the same as in the instance of INDEPENDENT SET.

For any $v \in V$, the set E_v can be determined in $O(|E|)$ time, so $\mathscr{C}$ can be constructed in $O(|V| \cdot |E|)$ time, which is polynomial in the size of the instance (G, k).

Now we show that $(\mathscr{C}, k)$ is a "yes" instance of SET PACKING if and only if (G, k) is a "yes" instance of INDEPENDENT SET.

Suppose $(\mathscr{C}, k)$ is a "yes" instance of SET PACKING. Then $\mathscr{C}$ contains k disjoint sets of the form $\{v_1\} \cup E_{v_1}$, $\{v_2\} \cup E_{v_2}$, …, $\{v_k\} \cup E_{v_k}$. Because these sets are disjoint, for $i \neq j$ no edge is in both E_{v_i} and E_{v_j}, which is to say that no edge is incident upon both v_i and v_j, so v_i and v_j are not adjacent. Therefore $\{v_1, v_2, \ldots, v_k\}$ is an independent set of size k in G, so (G, k) is a "yes" instance of INDEPENDENT SET.

Conversely, suppose (G, k) is a "yes" instance of INDEPENDENT SET. Then G contains an independent set $\{v_1, v_2, \ldots, v_k\}$ of size k. The corresponding sets $\{v_1\} \cup E_{v_1}$, $\{v_2\} \cup E_{v_2}$, …, $\{v_k\} \cup E_{v_k}$ in $\mathscr{C}$ must be disjoint, because for $i \neq j$ plainly $v_i \neq v_j$ and also $E_{v_i} \cap E_{v_j} = \emptyset$

because no edge is incident upon both v_i and v_j. Moreover $\{v_i\} \cup E_{v_i}$ and $\{v_j\} \cup E_{v_j}$ are distinct, because the first set contains v_i but the second does not. So these are k disjoint sets in $\mathscr{C}$, which means $(\mathscr{C}, k)$ is a "yes" instance of SET PACKING.

We showed in class that INDEPENDENT SET is NP-complete, so this polynomial-time transformation from INDEPENDENT SET to SET PACKING establishes that SET PACKING is NP-hard. □