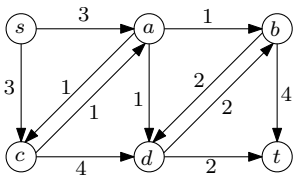
CSE 530: Algorithms & Complexity
Assignment 2: Suggested Answers

Antoine Vigneron

April 16, 2018

1. Find a minimum $s-t$ cut in the graph below. Justify your answer.



Answer. (See Figure 1.) The cut $(\{s, a, c, d\}, \{b, t\})$ has capacity 5. The flow below has value 5. So by the max-flow min-cut theorem, these are a minimum cut and a maximum flow.

2. A factory builds three types of radios A, B and C. Each radio is produced by the work of three specialists Pierre, Paul and Jacques. Pierre works at most 24 hours per week. Paul works at most 45 hours per week. Jacques works at most 30 hours per week. The resources necessary to build each type of radio and their selling prices are given in the following table:

	Radio A	Radio B	Radio C
Pierre	1h	2h	2h
Paul	2h	1h	2h
Jacques	1h	3h	1h
Selling price	\$15	\$10	\$12

We assume that the factory has no problem selling all the radios it produces. The goal is to maximize the revenue over one week.

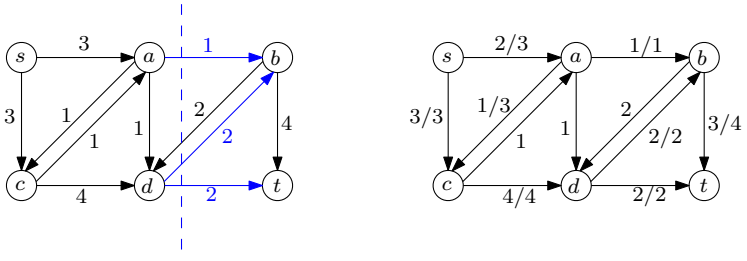


Figure 1: Solution to Problem 1.

(a) Formulate this problem as a linear program.

(b) Solve this linear program numerically using a linear program solver.

Answer a. We denote by a and b the numbers of radios of type A and B produced every week, respectively. The revenue generated each week is thus $15a + 10b$, which is the objective function of our linear program. Pierre works at most 24 hours per week, which gives the constraint $a + 2b \leqslant 24$. Similarly, we get the constraints $2a + b \leqslant 45$ and $a + 3b \leqslant 30$ for the work of Paul and Jacques. So we obtain the following linear program:

$$\begin{array}{lrcrcrl} \text{maximize} & 15a & + & 10b & + & 12c & \\ \text{subject to} & a & + & 2b & + & 2c & \leqslant 24 \\ & 2a & + & b & + & 2c & \leqslant 45 \\ & a & + & 3b & + & c & \leqslant 30 \\ & & & & & a, b, c & \geqslant 0 \end{array}$$

Answer b. We find $a = 22$, $b = 1$, $c = 0$. Value: $p = 340$

3. Consider the following problem:

$$\begin{array}{ll} \text{maximize} & x_1 + 2x_2 \\ \text{subject to} & |x_1| + |x_2| \leqslant 1. \end{array}$$

(a) Show that it is equivalent to a linear program.

(b) Solve this linear program by hand.

Answer. Since $|x_1| + |x_2| = \max(x_1 + x_2, x_1 - x_2, -x_1 - x_2, -x_1 + x_2)$, this problem is equivalent to the following linear program:

$$\begin{array}{ll} \text{maximize} & x_1 + 2x_2 \\ \text{subject to} & x_1 + x_2 \leqslant 1, \\ & x_1 - x_2 \leqslant 1, \\ & -x_1 - x_2 \leqslant 1, \\ & -x_1 + x_2 \leqslant 1. \end{array}$$

So we can solve it graphically as we did for the example in Lecture 2. The optimal point is $x^* = (0, 1)$, and the optimal value is 2. (See Figure 2.)

Another possible solution: For any feasible point (x_1, x_2), we have

$$\begin{array}{lll} x_1 + 2x_2 \leqslant |x_1| + 2|x_2| & & \\ \leqslant |x_1| + |x_1| + |x_2| & & \\ \leqslant |x_1| + 1 & & \text{because of the constraint } |x_1| + |x_2| \leqslant 1 \end{array}$$

But since $|x_1| + |x_2| \leqslant 1$, we also have $|x_1| \leqslant 1$. So $x_1 + 2x_2 \leqslant 2$. As the point $(0, 1)$ achieve this value of the objective function, it is an optimal solution to our problem.

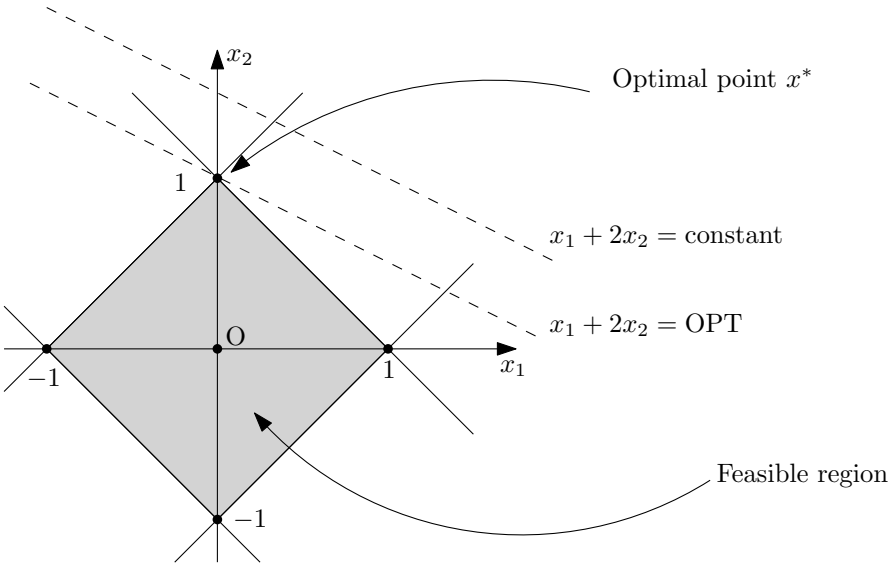


Figure 2: Graphical solution to Question 1.

4. We consider a generalized version of the maximum flow problem, where, in addition to the skew symmetry, flow conservation, and capacity constraint for each edge, we add a capacity constraint for each vertex (except s and t): The total amount of flow entering a vertex v is at most $c(v)$, where $c(v)$ is a given capacity. So a flow obeys the following constraints:

- $\forall u, v \in V,\ f(u,v) \leqslant c(u,v)$. (*Edge capacity constraint*)
- $\forall v \in V \setminus \{s,t\},\ \sum_{u \in V : f(u,v) > 0} f(u,v) \leqslant c(v)$. (*Vertex capacity constraint*)
- $\forall u, v \in V,\ f(u,v) = -f(v,u)$. (*Skew symmetry*)
- $\forall u \in V \setminus \{s,t\},\ \sum_{v \in V} f(u,v) = 0$. (*Flow conservation*)

Give an $O(|V|.|E|^2)$-time algorithm for the problem of computing a maximum flow in a network $G(V,E)$ of this type.

Answer. We will show how to reformulate any instance of maximum flow with vertex capacity as an ordinary instance of maximum flow, without vertex capacity. This new instance will have less than $2|V|$ vertices and $|E| + |V|$ edges. Thus, we can run the Edmonds-Karp algorithm on this new network, and obtain an optimal solution in $O(|V|.|E|^2)$ time.

Let v be a node with capacity $c(v)$. We replace it by two nodes v^+ and v^- in the new network without vertex capacity. We insert the edge (v^-, v^+) into the network, and we give it capacity $c(v^-, v^+) = c(u)$. (See Figure 3.) Then any incoming edge (u,v) to v is replaced with an incoming edge (u, v^-) and any outgoing edge (v,w) from v is replaced with an outgoing edge (v^+, w) from v. We apply this modification successively to each vertex of the original network, and we obtain a network without vertex capacity. It is easy to see that this new network is equivalent to the original.

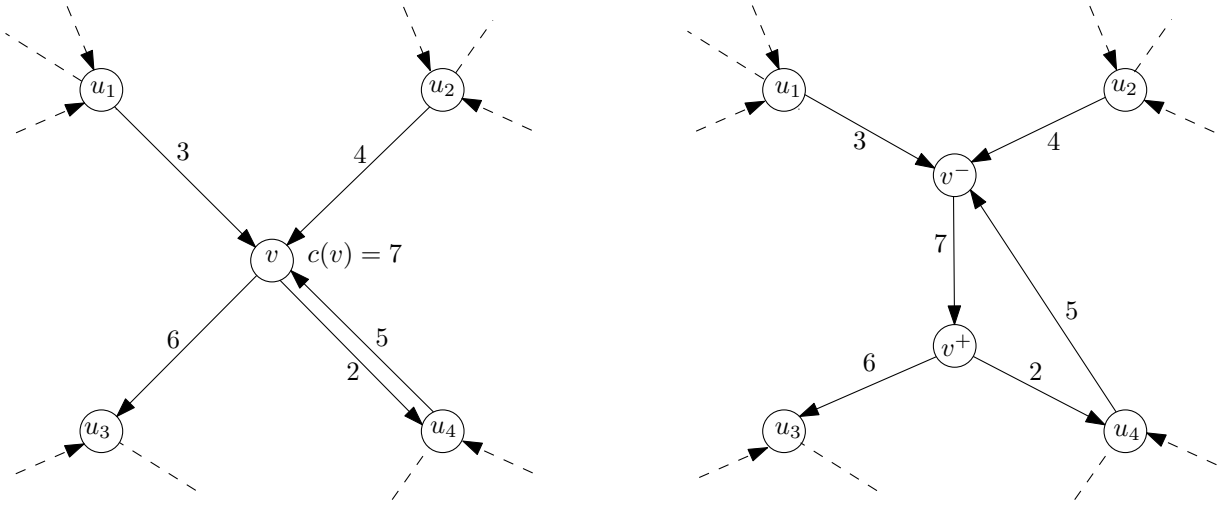


Figure 3: A flow network with vertex capacity (left), and an equivalent network without vertex capacity (right).

(There was a typo in the original problem statement, at the 2nd formula which expresses the vertex capacity constraint. On the other hand, the formulation in plain English that mentioned the total amount of flow *entering* the vertex was correct. the problem statement is now fixed.)

5. You are given a rectilinear polygon whose vertices have integer coordinates. (See Figure 4.) Give a polynomial-time algorithm to decide whether this polygon can be partitioned into 2×1 and 1×2 rectangles. When the answer is positive, your algorithm should also return one such partition.

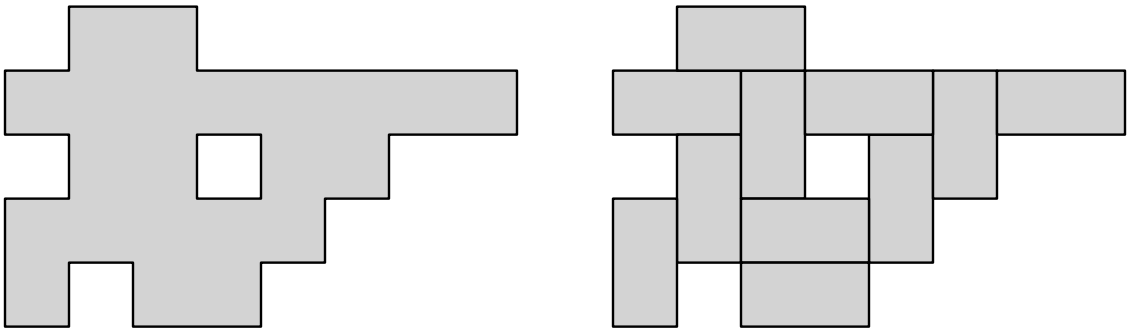


Figure 4: The input polygon (left) and its partition into 2×1 and 1×2 rectangles (right).

Answer. The idea is to reduce the problem to bipartite matching, which can be solved in polynomial time. (See Lecture 1.) The input polygon can be partitioned into unit squares with integer coordinates, that is, squares $s_i = [x_i, x_i + 1] \times [y_i, y_i + 1]$ where $x_i, y_i \in \mathbb{Z}$. Let S denote this set of squares. We partition it into two subset W, B where $s_i \in W$ if $x_i + y_i$ is even and $s_i \in B$

otherwise. Intuitively, we partitioned the polygon into black and white squares, as in a chessboard. (See Figure 5.)

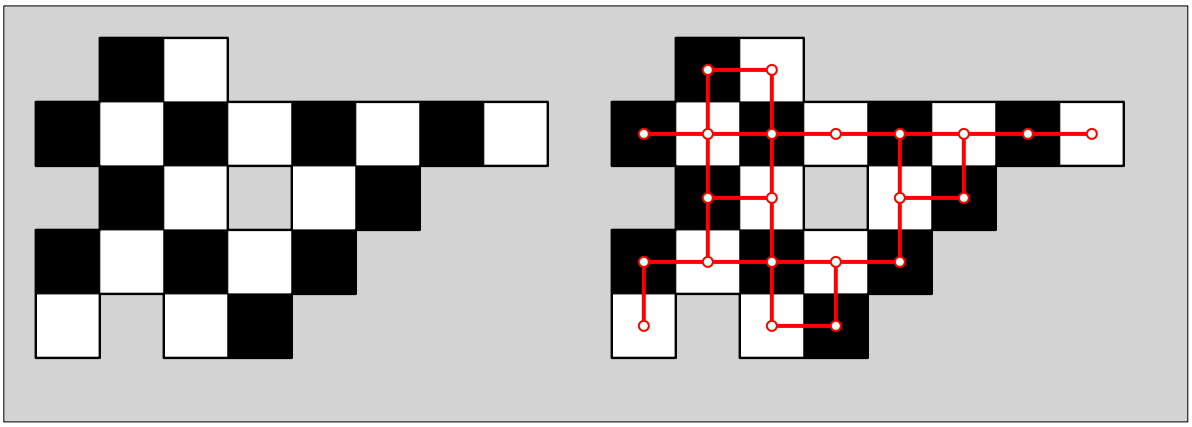


Figure 5: The partition into unit squares (left) and the corresponding bipartite graph (right).

Then we construct a graph G with vertex set S, where two squares are connected if they are adjacent along an edge. (See Figure 5.) This graph is bipartite because two white squares cannot be adjacent, and two black squares cannot be adjacent. Then a perfect matching in G corresponds to a partition of the polygon into dominoes, where each domino is the union of two squares connected by an edge in the matching. (See Figure 6.) So we just need to run a perfect matching algorithm on G: If there is no perfect matching, then no partition into dominoes exists, and otherwise the algorithm returns a matching and we return the corresponding partition.

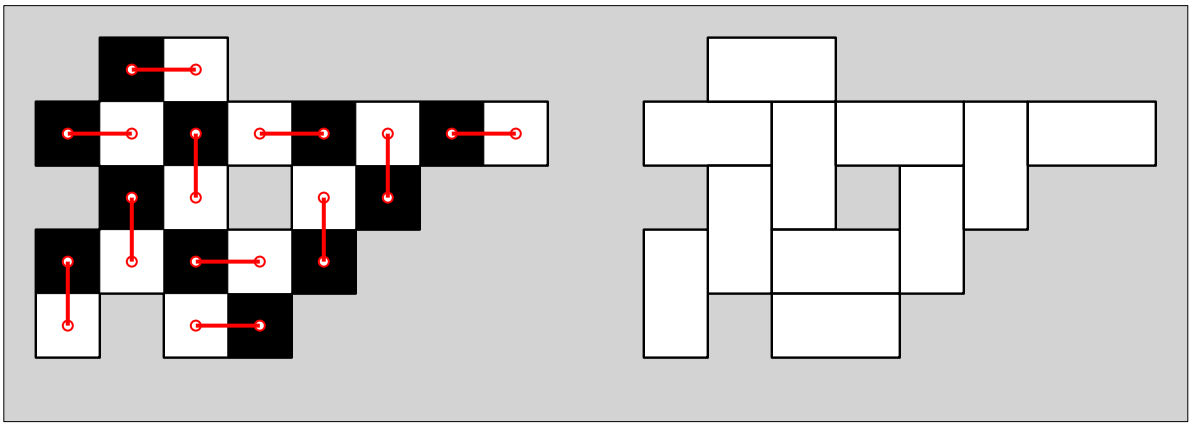


Figure 6: A perfect matching in G and the corresponding partition.