Approximation Algorithms Based on the Primal-Dual Method

The *primal-dual method* (or primal-dual schema) is another means of solving linear programs. The basic idea of this method is to start from a feasible solution $\mathbf{y}$ to the dual program, then attempt to find a feasible solution $\mathbf{x}$ to the primal program that satisfies the complementary slackness conditions. If such an $\mathbf{x}$ cannot be found, it turns out that we can find a better $\mathbf{y}$ in terms of its objective value. Then, another iteration is started.

The above idea can also be modified to design approximation algorithms. An approximate solution to the primal IP and a feasible solution to the dual LP can be constructed simultaneously and improved step by step. In the end, the approximate solution can be compared with the dual feasible solution to estimate the approximation ratio. One of the key strengths of this method is that it often allows for a combinatorial algorithm (based on the primal/dual view) which is very efficient.

1 Motivations: LP-algorithms for VERTEX COVER and SET COVER

Recall the (unweighted) VERTEX COVER problem. Given a graph $G = (V, E)$, taking all vertices of a maximal matching of G would give a vertex cover for G. This is a very efficient 2-approximation algorithm for the vertex cover problem.

The above algorithm runs much faster than the rounding algorithm we have seen. One might wonder how this algorithm looks from the angle of linear programming. The answer is a surprisingly nice one. Let us consider the linear relaxation of the integer program for vertex cover:

$$\begin{array}{ll} \min & \sum_{v\in V} x_v \\ \text{subject to} & x_u + x_v \geq 1, \quad \forall uv \in E, \\ & x_v \geq 0, \quad \forall v \in V. \end{array} \tag{1}$$

The dual program is

$$\begin{array}{ll} \max & \displaystyle\sum_{uv\in E} y_{uv} \\ \text{subject to} & \displaystyle\sum_{u:\ uv\in E} y_{uv} \leq 1, \quad \forall v \in V, \\ & y_{uv} \geq 0, \quad \forall uv \in E. \end{array} \tag{2}$$

An integral feasible solution to (2) corresponds to a matching of G. Based on the idea of a maximal matching in the view of this linear program, we define a feasible solution $\mathbf{y}$ to be *maximal* if there is no feasible solution $\mathbf{y}'$ for which $y'_{uv} \geq y_{uv}, \forall uv \in E$, and $\sum_{uv\in E} y'_{uv} > \sum_{uv\in E} y_{uv}$.

Now that we had the linear programming semantics of a maximal matching, the next question is: what does it mean to take both vertices of the maximal matchings? Easy, this corresponds to setting $x_v = 1$ whenever $\sum_{u:\ uv\in E} y_{uv} = 1$.

Theorem 1.1. *Let $\bar{\mathbf{y}}$ be a maximal feasible solution to* (2)*, then the strategy of setting $x_v^A = 1$ whenever $\sum_{u:\ uv\in E} \bar{y}_{uv} = 1$ gives a* 2*-approximation to the* VERTEX COVER *problem.*

Proof. We first show that $\mathbf{x}^A$ indeed defines a feasible vertex cover. If there is an edge $uv \in E$ such that both x_u^A and x_v^A are 0, then we must have

$$\sum_{s:\ us \in E} \bar{y}_{us} < 1, \ \ and \ \ \sum_{t:\ tv \in E} \bar{y}_{tv} < 1.$$

But then $\bar{y}_{uv}$ can be increased by an amount of

$$\delta = \min \left\{ \left(1 - \sum_{s:\ us \in E} \bar{y}_{us} \right), \left(1 - \sum_{t:\ tv \in E} \bar{y}_{tv} \right) \right\},$$

contradicting the maximality of $\bar{\mathbf{y}}$.

Secondly, we need to verify the approximation ratio of 2. This could be done easily using weak duality. We know $\bar{\mathbf{y}}$ gives a lower bound on the optimal value for (1), which is a lower bound of the optimal vertex cover:

$$\sum_v x_v^A \leq \sum_v \sum_{u:\ uv \in E} \bar{y}_{uv} = 2 \sum_{uv \in E} \bar{y}_{uv} \leq 2 \cdot \text{OPT}.$$

□

This algorithm can be extended easily to the weighted case, while the matching idea does not extend that well. Instead of solving the weighted case, let us see how this idea can be extended to the WEIGHTED SET COVER.

The integer program for WEIGHTED SET COVER is as follows.

$$\begin{aligned} \min \quad & w_1 x_1 + \cdots + w_n x_n \\ \text{subject to} \quad & \sum_{j: S_j \ni i} x_j \geq 1, \quad \forall i \in [m], \\ & x_j \in \{0, 1\}, \quad \forall j \in [n]. \end{aligned} \tag{3}$$

We assume the weights w_j to be non-negative. An LP relaxation for (3) is

$$\begin{aligned} \min \quad & w_1 x_1 + \cdots + w_n x_n \\ \text{subject to} \quad & \sum_{j: S_j \ni i} x_j \geq 1, \quad \forall i \in [m], \\ & x_j \geq 0, \quad \forall j \in [n]. \end{aligned} \tag{4}$$

We then have the dual program for (4):

$$\begin{aligned} \max \quad & y_1 + \cdots + y_m \\ \text{subject to} \quad & \sum_{i \in S_j} y_i \leq w_j, \quad \forall j \in [n], \\ & y_i \geq 0, \quad \forall i \in [m]. \end{aligned} \tag{5}$$

Combinatorially, to each set S_j we associate a non-negative number x_j and to each element i we associate a non-negative number y_i. The primal constraints say that the sum of numbers corresponding to sets containing an element i is at least one. The dual constraints say that the sum of numbers corresponding to elements in a set S_j is at most the weight w_j of the set. A feasible solution $\bar{\mathbf{y}}$ for (5) is said to be *maximal* if there is no other feasible solution $\mathbf{y}'$ for which $y_i' \geq \bar{y}_i, \forall i \in [m]$, and $\sum y_i' > \sum \bar{y}_i$.

Theorem 1.2. *Let $\bar{\mathbf{y}}$ be a maximal feasible solution to* (5)*, then the strategy of setting $x_j^A = 1$ whenever $\sum_{i \in S_j} y_i = w_j$ gives an f-approximation to the* WEIGHTED SET COVER *problem, where*

$$f = \max_i |\{j \mid i \in S_j\}|.$$

Proof. It's easy to see that x^A is feasible can be shown in the same manner as Theorem 1.1. As for the approximation ratio, we have

$$\begin{aligned}\sum_{j=1}^{n} w_j x_j^A &\leq \sum_{j=1}^{n} \sum_{i \in S_j} y_i \\ &= \sum_{i=1}^{m} |\{j : S_j \ni i\}| y_i \\ &\leq f \sum_{i=1}^{m} y_i \\ &\leq f \cdot \text{OPT}.\end{aligned}$$

□

Note that in this and the previous algorithms we wanted a maximal dual feasible solution $\bar{\mathbf{y}}$. One way to get a maximal dual feasible solution is to solve the dual LP. An optimal solution is certainly maximal.

2 The primal-dual method applied to SET COVER

We can get away with solving the dual LP. All we wanted was a maximal dual feasible solution, while solving the dual LP takes quite a bit of running time. In this section we present several approaches to get such a solution. In fact, we will not even need to explicitly compute the maximal dual feasible solution at all.

Let us consider the dual LP (5) for the WEIGHTED SET COVER problem. Certainly $\mathbf{y} = \mathbf{0}$ is a feasible solution. One way to obtain $\bar{\mathbf{y}}$ is to find an appropriate component y_i of $\mathbf{y}$ and increase it as much as possible to turn one more of the inequalities into equality. When this is not possible anymore, we get a maximal feasible solution for (5). The following algorithm implements this idea.

WSC-PRIMAL-DUAL-A

1: $\mathbf{y} \leftarrow 0$ // start with a feasible solution
2: $J \leftarrow \emptyset$ // Nothing yet in the cover
3: $I \leftarrow \{1, \ldots, m\}$ // Elements yet to be covered
4: **while** $I \neq \emptyset$ **do**
5: Pick $i \in I$ // and now try to increase y_i as much as possible
6: $j_i \leftarrow \arg\min\{w_j \mid j \in [n], i \in S_j\}$
7: $y_i \leftarrow w_{j_i}$ // this is the most y_i could be increased to
8: // the dual constraint corresponding to j_i shall become "binding" (becomes equality)
9: **for** each j where $i \in S_j$ **do**
10: $w_j \leftarrow w_j - w_{j_i}$
11: **end for**
12: $J \leftarrow J \cup \{j_i\}$
13: $I \leftarrow I - S_{j_i}$ // those in S_{j_i} are already covered
14: **end while**
15: Return J

The idea of the algorithm is very clear. We only need to show that the solution y it computes is feasible and maximal. Feasibility is maintained throughout as an invariant. To see maximality, consider the solution $\bar{\mathbf{y}}$ which was returned by the algorithm. For each $\bar{y}_i$, there is some S_j which contains i and

the dual constraint corresponding to j is binding. Thus, $\bar{y}_i$ cannot be increased without violating that constraint.

Note that we did not have to compute $\mathbf{y}$ at all. In algorithm WSC-PRIMAL-DUAL-A, removing all occurrences of $\mathbf{y}$ would not effect its result. The algorithm runs in $O(mn)$, which is very fast.

Exercise 1. Consider the following primal-dual algorithm to compute an approximate solution to the WEIGHTED SET COVER problem:

WSC-PRIMAL-DUAL-B

1: $J \leftarrow \emptyset$
2: $I \leftarrow \{1, \ldots, m\}$
3: $k \leftarrow 0$
4: **while** $I \neq \emptyset$ **do**
5: $\quad j_k \leftarrow \arg\min \left\{ \frac{w_j}{|S_j|} : j \in [n] \right\}$ // NOTE: the fraction is ∞ if $S_j = \emptyset$
6: $\quad S \leftarrow S_{j_k};\ r \leftarrow \frac{w_{j_k}}{|S|};\ J \leftarrow J \cup \{j_k\};\ I \leftarrow I - S$
7: $\quad$ **for** $j \leftarrow 1$ **to** n **do**
8: $\quad\quad w_j \leftarrow w_j - |S_j \cap S| r$
9: $\quad\quad S_j \leftarrow S_j - S$
10: $\quad$ **end for**
11: **end while**
12: Return J

(i) In words, explain what the algorithm does.

(ii) Show that the algorithm is an f-approximation algorithm to the WEIGHTED SET COVER problem, where f is the usual suspect.

Exercise 2. Consider the following integer program, which is equivalent to WEIGHTED SET COVER:

$$\begin{aligned} \min \quad & w_1 x_1 + \cdots + w_n x_n \\ \text{subject to} \quad & \sum_{j=1}^{n} |S_j \cap S| x_j \geq |S|, \quad \forall S \subseteq [m], \\ & x_j \in \{0, 1\}, \quad \forall j \in [n]. \end{aligned} \tag{6}$$

(i) Prove that this program is equivalent to the WEIGHTED SET COVER problem. Equivalence means every optimal solution to WEIGHTED SET COVER corresponds to an optimal solution of (6) and vice versa.

(ii) Write down its LP relaxation and the LP's dual program.

(iii) Design an approximation algorithm using the primal-dual method similar to the one presented in the lecture. Your algorithm should have approximation factor f, as usual.

You should briefly describe the idea and write down the pseudo code. Also, what's the running time of your method?

Exercise 3. Consider the HITTING SET problem, in which we are given a universe set U with weighted elements, and a collection of subsets $T_1, \ldots, T_k$ of U. The problem is to find a minimum-weight subset A of U which hits every T_i, namely $A \cap T_i \neq \emptyset$, for all $i \in [k]$.

- Write an integer program formulation of this problem. Also write the dual linear program of the relaxed version of the IP.

- Devise a primal-dual approximation algorithm for this problem.
- What's your approximation ratio and running time?

3 An LP-algorithm for the GENERAL COVER problem

The integer program for the GENERAL COVER problem has the following form

$$\begin{array}{lccccccll} \min & c_1x_1 & + & \ldots & + & c_nx_n & & & \\ \text{subject to} & a_{i1}x_1 & + & \ldots & + & a_{in}x_n & \geq & b_i, & \forall i \in [m]. \\ & & & & & & & x_j \in \{0,1\}, & \forall j \in [n], \end{array} \tag{7}$$

where a_{ij}, b_i, c_j are all non-negative integers. Since we can remove an inequality if $b_i = 0$, we can assume that $b_i > 0, \forall i \in [n]$. Moreover, if $c_j = 0$ then we can set $x_j = 1$ and remove the column corresponding to j without effecting the objective function as well as feasible solutions. Thus, we can also assume that $c_j > 0, \forall j \in [n]$. Lastly, for each row i we must have $\sum_j a_{ij} \geq b_i$, otherwise the problem is clearly infeasible.

The relaxed LP version for (7) is

$$\begin{array}{lccccccll} \min & c_1x_1 & + & \ldots & + & c_nx_n & & & \\ \text{subject to} & a_{i1}x_1 & + & \ldots & + & a_{in}x_n & \geq & b_i, & \forall i \in [m]. \\ & & & & & & & 0 \leq x_j \leq 1, & \forall j \in [n], \end{array} \tag{8}$$

Note that we **cannot** remove the $x_j \leq 1$ constraints as in WEIGHTED VERTEX COVER and WEIGHTED SET COVER counterparts (why?).

As before, the rounding approach for this problem requires solving a linear program (either the dual or the primal), which has large running time in the worst case. We attempt here to develop more efficient algorithms to solve this problem.

Let's first write down the dual LP of (8). To each constraint $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ we associate a dual variable y_i, and to each constraint $-x_j \geq -1$ we associate a variable z_j. The dual LP is then

$$\begin{array}{lccccll} \max & b_1y_1 + \cdots + b_my_m - (z_1 + \cdots + z_n) & & & & & \\ \text{subject to} & a_{1j}y_1 + \cdots + a_{mj}y_m - z_j & \leq & c_j, & \forall j \in [n]. \\ & y_i, z_j & \geq & 0, & \forall i \in [m], j \in [n] \ , \end{array} \tag{9}$$

Each feasible solution to (9) has the form $(\mathbf{y}, \mathbf{z})$. (To be very rigorous, we should have written $(\mathbf{y}^T, \mathbf{z}^T)^T$.) In order to mimic the LP-algorithm for WEIGHTED VERTEX COVER and WEIGHTED SET COVER, we first need to define what a maximal solution to (9) is. The concept of a maximal solution $(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ should be defined so that, if we set

$$x_j^A = 1, \ \text{ whenever } \ \sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j = c_j, \tag{10}$$

then we get a feasible solution $\mathbf{x}^A$ to the primal program. Let us analyze what kind of $(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ would make $\mathbf{x}^A$ feasible. Let

$$J := \left\{ j \mid x_j^A = 1 \right\} = \left\{ j \mid \sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j = c_j \right\}.$$

Suppose after the assignment (10) $\mathbf{x}^A$ is not feasible. This means there is some row i for which

$$\sum_{j \in J} a_{ij} < b_i.$$

Since we have assumed that $\sum_j a_{ij} \geq b_i$, it must be the case that $J \neq [n]$. Let $\bar{J} = [n] - J \neq \emptyset$. In order to fix this miscue, we must turn one or more $x_j^A, j \in \bar{J}$ to be 1. In other words, we want one or more $j \in \bar{J}$ such that

$$\sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j = c_j.$$

This can be done by increasing $\bar{y}_i$ by some small amount δ so as to turn one or more of the following inequalities

$$\sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j < c_j, \;\; j \in \bar{J}$$

into equality. Firstly, to keep the feasibility of $(\bar{y}, \bar{z})$, the equalities corresponding to $j \in J$ must still be maintained. This is easy as for $j \in J$, we have

$$c_j = \sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j = \sum_{i=1}^{m} a_{ij}\bar{y}_i + a_{ij}\delta - (\bar{z}_j + a_{ij}\delta).$$

In other words, we only need to increase $\bar{z}_j$ by $a_{ij}\delta$ to maintain those equalities. The number δ can be determined by

$$\delta = \min\left\{ \frac{c_j - (\sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j)}{a_{ij}} \mid j \in \bar{J}, a_{ij} > 0 \right\}.$$

(This δ is well defined as $\sum_{j \in \bar{J}} a_{ij} > 0$.) After this increase, the objective function of the dual program is increased by

$$b_i\delta - \sum_{j \in J} a_{ij}\delta,$$

which is a positive amount.

In this entire process, we have increased the vector $(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ and also the objective function of the dual program. Hence, if $(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ were to be maximal in the sense that there is no other feasible vector $(\mathbf{y}, \mathbf{z})$ which gives strictly larger dual objective value and $y_i \geq \bar{y}_i, \forall i$, and $z_j \geq \bar{z}_j, \forall j$, then x^A has to be feasible.

For our purposes, the definition above of the maximal dual feasible solution is not sufficient yet. It must satisfy another property for the analysis of the approximation ratio (our favorite f) to go through. We want, as in the WEIGHTED SET COVER case,

$$c_j x_j^A \leq \sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j. \tag{11}$$

This is certainly true when $j \in J$, but it may not be true when $j \in \bar{J}$, i.e. when $\sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j < c_j$. We need to maintain the property that $\sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j \geq 0$ for this analysis to go through.

To this end, let's assume that (11) holds true for all j. We have

$$\begin{aligned}
\sum_{j=1}^{n} c_j x_j^A &\leq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij}\bar{y}_i - \bar{z}_j \right) \\
&\leq \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} \right) \bar{y}_i \\
&\leq f \sum_{i=1}^{m} \bar{y}_i.
\end{aligned}$$

In the WEIGHTED SET COVER case, the sum $\sum_{i=1}^{m} \bar{y}_i$ was precisely the objective value of the dual feasible solution $\bar{y}$, which was at most the optimal value of WEIGHTED SET COVER. Here, the objective value of the dual feasible solution $(\bar{y}, \bar{z})$ is $\sum_{i=1}^{m} \bar{y}_i b_i - \sum_{j=1}^{n} \bar{z}_j$. Thus, for this proof to go through we also want

$$\sum_{i=1}^{m} \bar{y}_i \leq \sum_{i=1}^{m} \bar{y}_i b_i - \sum_{j=1}^{n} \bar{z}_j.$$

If that was the case, we can continue with the analysis:

$$\begin{aligned} \sum_{j=1}^{n} c_j x_j^A &\leq f \left(\sum_{i=1}^{m} \bar{y}_i b_i - \sum_{j=1}^{n} \bar{z}_j \right) \\ &\leq f \cdot \text{OPT}. \end{aligned}$$

The analysis above leads to the following definition and theorem.

Definition 3.1 (Maximal dual feasible solution)**.** A feasible solution $(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ of (9) is said to be *maximal* if it satisfies

1. There is no other feasible solution (y, z) for which $y \geq \bar{y}$, $z \geq \bar{z}$, and $\sum_{i=1}^{m} y_i b_i - \sum_{j=1}^{n} z_j > \sum_{i=1}^{m} \bar{y}_i b_i - \sum_{j=1}^{n} \bar{z}_j$.
2. $\sum_{i=1}^{m} a_{ij} \bar{y}_i - \bar{z}_j \geq 0$, for all $j \in [n]$.
3. $\sum_{i=1}^{m} \bar{y}_i \leq \sum_{i=1}^{m} \bar{y}_i b_i - \sum_{j=1}^{n} \bar{z}_j$.

Theorem 3.2. *Suppose* $(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ *is a maximal feasible solution to* (9)*, then the strategy of setting*

$$x_j^A = 1, \ \textit{whenever} \ \sum_{i=1}^{m} a_{ij} \bar{y}_i - \bar{z}_j = c_j$$

gives an f *approximation algorithm to the* GENERAL COVER *problem, where* $f = \max_i \sum_j a_{ij}$.

The following exercise shows one of the ways to find a maximal solution.

Exercise 4. Let $(\mathbf{y}^*, \mathbf{z}^*)$ be an optimal solution to (9), is it true that $(\mathbf{y}^*, \mathbf{z}^*)$ is maximal? Explain your conclusion.

4 A primal-dual algorithm for GENERAL COVER

The analysis provided in the previous section actually gives rise to an efficient algorithm which finds a dual feasible solution, in much the same way as the primal-dual algorithms for WEIGHTED VERTEX COVER and WEIGHTED SET COVER. We start from $(\mathbf{y}, \mathbf{z}) = (0, 0)$, which is certainly feasible and maintain the following conditions as invariants (beside the obvious ones $y_i \geq 0, z_j \geq 0$):

$$c_j \geq \sum_{i=1}^{m} a_{ij} y_i - z_j \geq 0, \forall j \in [n], \tag{12}$$

and

$$\sum_{j=1}^{n} z_j \leq \sum_{i=1}^{m} y_i (b_i - 1). \tag{13}$$

The algorithm stops when the first criterion for the maximality of $(\mathbf{y}, \mathbf{z})$ is reached. We first give a slow version, which is more convenient to prove the algorithm's correctness.

GC-PRIMAL-DUAL-A1

1: $\mathbf{y} \leftarrow 0$, $\mathbf{z} \leftarrow 0$ // start with a feasible solution
2: $J \leftarrow \emptyset$ // nothing in the cover yet
3: $I \leftarrow \{1, \ldots, m\}$ // rows yet to be covered (satisfied)
4: **while** $I \neq \emptyset$ **do**
5: Pick $i \in I$ // and now try to increase y_i as much as possible
6: $\bar{j} \leftarrow \arg\min \left\{ \frac{c_j - \sum_{i=1}^{m} a_{ij} y_i + z_j}{a_{ij}} \mid j \in \bar{J}, a_{ij} > 0 \right\}$
7: $\delta \leftarrow \min \left\{ \frac{c_j - \sum_{i=1}^{m} a_{ij} y_i + z_j}{a_{ij}} \mid j \in \bar{J}, a_{ij} > 0 \right\}$ // δ could be 0 in a few iterations
8: $y_i \leftarrow y_i + \delta$ // this is the most y_i could be increased to
9: // the dual constraint corresponding to $\bar{j}$ shall become "binding" (becomes equality)
10: **for** each $j \in J$ **do**
11: $z_j \leftarrow z_j + a_{ij}\delta$ // maintain the fact that $\sum_{i=1}^{m} a_{ij} y_i - z_j = c_j, \forall j \in J$
12: **end for**
13: $J \leftarrow J \cup \{\bar{j}\}$ // or $x_{\bar{j}}^{A} \leftarrow 1$ for that matter
14: $R \leftarrow \{r \in I \mid \sum_{j \in J} a_{rj} \geq b_r\}$ // R may be empty in some iteration
15: $I \leftarrow I - R$ // constraints corresponding to R are satisfied
16: **end while**
17: Return J

Theorem 4.1. *Algorithm* GC-PRIMAL-DUAL-A1 *computes a maximal dual feasible solution* $(\mathbf{y}, \mathbf{z})$*, and returns a solution* J *which gives rise to an* f*-approximation algorithm for the* GENERAL COVER *problem.*

Proof. We first show that conditions (12) and (13) are invariant throughout the execution of the algorithm. If you want to be extremely rigorous, an inductive proof could be used, but the algorithm needs to be rewritten a little bit to accommodate such proof.

The fact that $\sum_{i=1}^{m} a_{ij} y_i - z_j \leq c_j$, for $j \in \bar{J}$, trivially holds as δ was carefully chosen to keep it true. When $j \in J$, $\sum_{i=1}^{m} a_{ij} y_i - z_j = c_j$ before the increase of y_i in the iteration and it is obviously still true after each iteration as we have increased z_j by an appropriate amount.

To see that $\sum_{i=1}^{m} a_{ij} y_i - z_j \geq 0$, we only have to notice that $z_j = 0$ for all $j \in \bar{J}$. Hence, when $j \in J$ we have $\sum_{i=1}^{m} a_{ij} y_i - z_j = c_j \geq 0$, while if $j \in \bar{J}$ we have $\sum_{i=1}^{m} a_{ij} y_i - z_j = \sum_{i=1}^{m} a_{ij} y_i \geq 0$.

Next, assume that (13) holds before each iteration; we shall show that it still holds after each iteration. The left hand side of (13) is increased by $\sum_{j \in J} a_{ij}\delta$, while the right hand side was increased by $(b_i - 1)\delta$. Before each iteration, I consists precisely of the rows r with $\sum_{j \in J} a_{rj} \leq b_r - 1$. In particular $\sum_{j \in J} a_{ij} \leq b_i - 1$, yielding the desired result.

Lastly, we check the maximality of the final solution $(\bar{\mathbf{y}}, \bar{\mathbf{z}})$. Suppose there is another dual feasible solution $(\mathbf{y}, \mathbf{z})$ for which $\mathbf{y} \geq \bar{\mathbf{y}}$ and $\mathbf{z} \geq \bar{\mathbf{z}}$. Let J be the set returned by the algorithm. The idea is that increasing any of the y_i would force too much increase on the z_j, $j \in J$, to the point that the objective

value is reduced. In equations, we have

$$\begin{aligned}
\sum_{i=1}^{m}(y_i-\bar{y}_i)b_i-\sum_{j=1}^{n}(z_j-\bar{z}_j) &\leq \sum_{i=1}^{m}(y_i-\bar{y}_i)\left(\sum_{j\in J}a_{ij}\right)-\sum_{j=1}^{n}(z_j-\bar{z}_j)\\
&= \sum_{j\in J}\sum_{i=1}^{m}a_{ij}(y_i-\bar{y}_i)-\sum_{j=1}^{n}(z_j-\bar{z}_j)\\
&= \sum_{j\in J}\left(\sum_{i=1}^{m}a_{ij}y_i-\sum_{i=1}^{m}a_{ij}\bar{y}_i\right)-\sum_{j=1}^{n}(z_j-\bar{z}_j)\\
&\leq \sum_{j\in J}\left(z_j+c_j-\sum_{i=1}^{m}a_{ij}\bar{y}_i\right)-\sum_{j=1}^{n}(z_j-\bar{z}_j)\\
&= \sum_{j\in J}(z_j-\bar{z}_j)-\sum_{j=1}^{n}(z_j-\bar{z}_j)\\
&= -\sum_{j\in\bar{J}}(z_j-\bar{z}_j)\\
&\leq 0.
\end{aligned}$$

□

Since we do not have to compute explicitly vector $(\mathbf{y},\mathbf{z})$, the previous algorithm can be rewritten as follows.

GC-PRIMAL-DUAL-A2

1: $J\leftarrow\emptyset$
2: $I\leftarrow\{1,\ldots,m\}$
3: **while** $I\neq\emptyset$ **do**
4: Pick $i\in I$ // and now try to increase y_i as much as possible
5: $\bar{j}\leftarrow\arg\min\left\{\frac{c_j}{a_{ij}}\mid j\in\bar{J},a_{ij}>0\right\}$
6: $\delta\leftarrow\min\left\{\frac{c_j}{a_{ij}}\mid j\in\bar{J},a_{ij}>0\right\}$
7: **for** each $j\in\bar{J}$ **do**
8: $c_j\leftarrow c_j-a_{ij}\delta$
9: **end for**
10: $J\leftarrow J\cup\{\bar{j}\}$
11: **for** each $r\in I$ **do**
12: $b_r\leftarrow b_r-a_{r\bar{j}}$
13: **end for**
14: **for** each $r\in I$ **do**
15: **if** $b_r\leftarrow 0$ **then**
16: $I\leftarrow I-\{r\}$
17: **end if**
18: **end for**
19: **end while**
20: Return J

Exercise 5. In words, explain how algorithm GC-PRIMAL-DUAL-A2 works as compared to algorithm GC-PRIMAL-DUAL-A1.

Exercise 6. Recall the following algorithm for WEIGHTED SET COVER:

WSC-PRIMAL-DUAL-B

1: $J \leftarrow \emptyset$
2: $I \leftarrow \{1, \ldots, m\}$
3: $k \leftarrow 0$
4: **while** $I \neq \emptyset$ **do**
5: $\quad j_k \leftarrow \arg\min \left\{ \frac{w_j}{|S_j|} : j \in [n] \right\}$ // NOTE: the fraction is ∞ if $S_j = \emptyset$
6: $\quad S \leftarrow S_{j_k};\ r \leftarrow \frac{w_{j_k}}{|S|};\ J \leftarrow J \cup \{j_k\};\ I \leftarrow I - S$
7: $\quad$ **for** $j \leftarrow 1$ **to** n **do**
8: $\quad\quad w_j \leftarrow w_j - |S_j \cap S| r$
9: $\quad\quad S_j \leftarrow S_j - S$
10: $\quad$ **end for**
11: **end while**
12: Return J

(i) Write the pseudo-code for an approximation algorithm which uses the same idea to solve the GENERAL COVER problem.

(ii) Prove the correctness of your algorithm.

5 A brief introduction to network flows

A *flow network* is a directed graph $D = (V, E)$ with two distinguished vertices s and t called the *source* and the *sink*, respectively. Moreover, each arc $(u, v) \in E$ has a certain *capacity* $c(u, v) \geq 0$ assigned to it. If $(u, v) \notin E$ (including pairs of the form (u, u)), we assume $c(u, v) = 0$. In this note, we shall restrict ourselves to the case where **capacities are all rational numbers**. Some of the capacities might be ∞.

Let $G = (V, E)$ be a graph or a digraph. Let X be a proper non-empty subset of V. Let $\bar{X} := V - X$, then the pair $(X, \bar{X})$ forms a partition of V, called a *cut* of G. The set of edges of G with one end point in each of X and $\bar{X}$ is called an *edge cut* of G, denoted by $[X, \bar{X}]$. When G is a directed graph, let

$$\begin{aligned} \delta^+(X) &= \{(u, v) \mid u \in X, v \notin x\}, \\ \delta^-(X) &= \{(u, v) \mid u \notin X, v \in x\}. \end{aligned}$$

A *source/sink cut* of a network D is a cut (S, T) with $s \in S$ and $t \in T$. (Note that, implicitly $T = \bar{S}$.)

A *flow* for a network $D = (V, E)$ is a function $f : V \times V \to \mathbb{R}$, which assigns a real number to each pair (u, v) of vertices. A flow f is called a *feasible flow* if it satisfies the following conditions:

(i) $0 \leq f(u, v) \leq c(u, v), \forall (u, v) \in E$. These are the *capacity constraints*. (If a capacity is ∞, then there is no upper bound on the flow value on that edge.)

(ii) For all $v \in V - \{s, t\}$, the total flow into v is the same as the total flow out of v, namely

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{w:(v,w)\in E} f(v, w). \tag{14}$$

These are called the *flow conservation law*.

The *value* of a flow f for D, denoted by $\text{val}(f)$, is the net flow out of the source:

$$\text{val}(f) := \sum_{u:(s,u)\in E} f(s,u) - \sum_{v:(v,s)\in E} f(v,s).$$

For notational conveniences, for every two subsets $X, Y \subseteq V$, define

$$f(X,Y) := \sum_{x\in X}\sum_{y\in Y} f(x,y).$$

For every proper and non-empty subset $S \subseteq V$ we define $f^+(S)$ to be the net flow leaving S and $f^-(S)$ to be the net flow entering S, namely

$$f^+(S) := f(S,\bar{S}), \tag{15}$$

$$f^-(S) := f(\bar{S},S). \tag{16}$$

If $S = \{w\}$ for some vertex $w \in V$, we write $f^+(w)$ and $f^-(w)$ instead of $f^+(\{w\})$ and $f^-(\{w\})$, respectively. The flow conservation law (14) now reads $f^+(v) = f^-(v), \forall v \in V - \{s,t\}$. And, the value of f is nothing but $f^+(s) - f^-(s)$.

With the conservation law held at all vertices other than the source and the sink, it is intuitively clear that the net flow into the sink is also $\text{val}(f)$.

Exercise 7. The value of a flow f is equal to the net flow into the sink:

$$\text{val}(f) = f^-(t) - f^+(t).$$

Definition 5.1 (The Maximum Flow problem)**.** The *maximum flow problem* is the problem of finding a feasible flow with maximum value, given a network D (and the capacities on the edges).

Exercise 8. Formulate the maximum flow problem as a linear program.

Given a source/sink cut (S,T), the *capacity* of the cut, denoted by $\text{cap}(S,T)$ is the total capacity of edges leaving S:

$$\text{cap}(S,T) := \sum_{\substack{u\in S, v\in T,\\ (u,v)\in E}} c(u,v).$$

A cut with minimum capacity is called a *minimum cut*.

Exercise 9. Given a source/sink cut (S,T) and a feasible flow f for a network D, show that

$$\text{val}(f) = f^+(S) - f^-(S) = f^-(T) - f^+(T).$$

Theorem 5.2 (Weak duality for flow networks)**.** *For every source/sink cut $[S,T]$ and any feasible flow f for a network $D = (V,E)$, we have*

$$\text{val}(f) \le \text{cap}(S,T).$$

Due to the weak duality property, a feasible flow f with value equal to the capacity of some cut $[S,T]$ is a *maximum flow*. The cut is then a *minimum cut*. Similar to linear programs, strong duality also holds for flow networks (which is a special case of linear programming). This fact is the content of the max-flow min-cut theorem.

Theorem 5.3 (Max-flow min-cut)**.** *Let f be a feasible flow of a network D, then f is a maximum flow if and only if there is some source/sink cut $[S,T]$ with* $\text{val}(f) = \text{cap}(S,T)$

Another fact crucial to many optimization problems is as follows.

Theorem 5.4 (Integrality theorem)**.** *If the finite capacities are all integers, and the maximum flow is bounded, then there is a maximum flow f in which $f(u,v)$ and* $\text{val}(f)$ *are all integers.*

6 Optimization problems formulated as SET COVER

There are quite a few optimization problems that can be formulated as a (weighted) SET COVER problem. To simplify notations, we will recap the SET COVER problem using a simpler set of notations.

Recall that in the SET COVER problem, we are given a universe set U, and a collection $\mathcal{C}$ of subsets of U. Implicitly, let $m = |U|$ and $n = |\mathcal{C}|$. Each set S in $\mathcal{C}$ is weighted with a non-negative integer weight w_S. The corresponding integer program is

$$\begin{array}{ll} \min & \sum_{S\in\mathcal{C}} w_S x_S \\ \text{subject to} & \displaystyle\sum_{S\ni i} x_S \geq 1, \quad \forall i \in U, \\ & x_S \in \{0,1\}, \quad \forall S \in \mathcal{C}. \end{array} \tag{17}$$

It turns out that many optimization problems follow the same setting. In the following problems, implicitly we assume all weights are non-negative integers.

Consider the SHORTEST s-t PATH PROBLEM, in which we need to find a shortest path between two vertices s and t of a graph $G = (V, E)$. Let the universe U be the set of all s-t cuts. For each edge $(u, v) \in E$, let S_{uv} be the set of all cuts in U that contain (u, v). Let $\mathcal{C}$ be the collection of all $S_{u,v}$. If E is weighted, then the weight of $S_{u,v}$ is the weight of (u, v). Then, by the max-flow min-cut theorem and the integrality theorem for flow networks, it is easy to see that SHORTEST s-t PATH is equivalent to the SET COVER problem on U and $\mathcal{C}$ (weighted or not). The basic idea is to pick a minimum-weight set of edges that "cover" all s-t cuts!

Similarly, in the MINIMUM SPANNING TREE problem we need a minimum-weight set of edges that cover all cuts in a graph G.

The GENERALIZED STEINER TREE problem is defined as follows. Given an edge-weighted undirected graph $G = (V, E)$ and m pairs of vertices (s_j, t_j), $j \in [m]$. Find a minimum-weight subset of edges $C \subseteq E$ such that s_j and t_j are connected in (V, C), for each $j \in [m]$. In this problem, we want C to cover all s_j-t_j cuts.

In the FEEDBACK VERTEX SET problem, we are given an undirected graph $G = (V, E)$ with weighted vertices. The goal is to find a minimum-weight subset C of vertices so that every cycle in G contains some vertex in C. Thus, we want C to cover all cycles of G.

The MINIMUM-COST ARBORESCENCE PROBLEM, also called the MINIMUM-COST BRANCHING PROBLEM is defined as follows. Given a directed, edge-weighted graph $G = (V, E)$ with a special vertex r called the root. Find a minimum-cost spanning tree where edges are directed away from r. In this case, we want the edges of the tree to cover all r-directed cuts.

Many of the problems above can be formulated with the following integer program. We assume that an edge-weighted graph $G = (V, E)$ is given, and edge e is weighted with $w_e \in \mathbb{Z}^+$.

$$\begin{array}{ll} \min & \sum_{e\in E} w_e x_e \\ \text{subject to} & \displaystyle\sum_{e\in\delta(X)} x_e \geq f(X), \quad \emptyset \neq X \subset V, \\ & x_e \in \{0,1\}, \qquad \forall e \in E. \end{array} \tag{18}$$

Here $\delta(X) = [X, \overline{X}]$, and $f : 2^V \to \mathbb{Z}^+$ is a function that counts how many edges must cross $\delta(X)$ in a feasible solution. In the SHORTEST s-t PATH problem, for instance, each s-t cut must be crossed at least once. Other problems follow the same trend, except for the FEEDBACK VERTEX SET problem.

A very general problem of this type is called the SURVIVABLE NETWORK DESIGN PROBLEM, also called the GENERALIZED STEINER PROBLEM. We are given an edge-weighted graph $G = (V, E)$. For each pair u, v of vertices, there is a non-negative integer m_{uv}. We must find a least-cost subset C of edges such that in (V, C) there are r_{uv} edge disjoint paths joining u and v, for each pair u, v. In this case, we want $f(X) = \max_{u\in X, v\notin X} m_{uv}$ for each subset X of vertices.

We shall see how the primal-dual method helps design approximation algorithms for problems formulated as (18) for several classes of functions f.

7 A closer look at the algorithm for SET COVER

The dual linear program of the relaxed version of (17) is

$$\begin{array}{ll} \max & \sum_{i\in U} y_i \\ \text{subject to} & \sum_{i\in S} y_i \le w_S, \quad \forall S \in \mathcal{C}, \\ & y_i \ge 0, \quad \forall i \in U. \end{array} \tag{19}$$

The basic primal-dual approximation algorithm for the SET COVER problem can be summarized as follows. PRIMAL-DUAL BASIC

1: $\mathbf{y} \leftarrow 0$
2: $C \leftarrow \emptyset$
3: **while** C is not a cover **do**
4: Choose an uncovered element k
5: Increase y_k until $\exists S : \sum_{i\in S} y_i = w_S$
6: Add S into C
7: **end while**
8: Return C (call it $\bar{C}$)

Recall that, from the linear programming view $x_S^A = 1$ only when $\sum_{i\in S} y_i = w_S$ (we say S is *saturated*). Note that there might be saturated sets S which are not in $\bar{C}$ ($x_S^A \neq 1$). Our analysis goes as follows

$$\text{cost}(x^A) = \sum_{S\in\bar{C}} w_S x_S^A = \sum_{S\in\bar{C}} \left(\sum_{i\in S} y_i \right) = \sum_{i\in U} |\{S : S \in \bar{C}, i \in S\}| y_i.$$

For any collection C of subsets of U, and any $i \in U$, let $g(C, i) = |\{S : S \in C, i \in S\}|$. If there was an α such that $g(\bar{C}, i) \le \alpha$ whenever $y_i > 0$, then we have

$$\text{cost}(x^A) = \sum_{i\in U} g(\bar{C}, i) y_i \le \alpha \sum_{i\in U} y_i \le \alpha \cdot \text{OPT}.$$

We thus have proved the following theorem.

Theorem 7.1. *Let $\bar{C}$ be the collection returned by* PRIMAL-DUAL BASIC. *Let α be any number such that, for each $i \in U$, $y_i > 0$ implies $g(\bar{C}, i) \le \alpha$. Then,* PRIMAL-DUAL BASIC *is an α-approximation algorithm for* SET COVER.

For the SET COVER problem, α can be chosen to be the maximum number of appearances of an element in the given collection. The situation is not as simple for other problems.

7.1 Be particular in choosing the uncovered element k

Consider the FEEDBACK VERTEX SET problem on a graph $G = (V, E)$. We want to cover the universe of cycles of G with vertices in V. For a collection of vertices $\bar{C}$ and some cycle i, $g(\bar{C}, i)$ is the number of vertices of $\bar{C}$ in the cycle. The rough bound is n, because a cycle may contain as many as n vertices. This is a very bad approximation ratio.

Fortunately, to obtain a ratio of α we only need $g(\bar{C}, i) \leq \alpha$ when $y_i > 0$, i.e. when cycle i is chosen at some point during the execution of PRIMAL-DUAL BASIC. Thus, if we try our best to pick small cycles, we would have a better approximation. Further more, not all vertices in a chosen cycle will be in $\bar{C}$ at the end. Hence, $g(\bar{C}, i)$ can still be smaller than the length of cycle i.

To this end, one needs to be creative to take advantage of the above observations. We will limit the set of vertices which have the potential to be in the final cover. We shall refer to them as *interesting* vertices. If we never pick a uninteresting vertex to be in the cover, then $g(\bar{C}, i)$ is the number of interesting vertices on cycle i, which can potentially be smaller than the size of this cycle.

All vertices of degree 1 are not interesting. If there was a path P of G, all of whose internal vertices have degree 2, then only a least-weight internal vertex v needs to be interesting among all internal vertices, because a cycle containing v will contain all of P. Consequently, if at each step of the algorithm we can somehow choose an uncovered cycle containing a small number of interesting vertices, then have would be able to improve the approximation ratio. A result of Erdős and Pósa in 1962 tells us how to do this.

Theorem 7.2 (Erdős-Pósa [7])**.** *Let $G' = (V', E')$ be a graph with no degree-1 vertex in which each degree-2 vertex is adjacent to two vertices of degrees at least 3. Then, G' has a cycle of length at most $4 \lg |V'|$. Moreover, this cycle can be found in polynomial time.*

Exercise 10. Consider a graph G with no degree-1 vertex in which each degree-2 vertex is adjacent to two vertices of degrees at least 3. Let H be the graph obtained from G by shortcutting all vertices of degree 2, namely for each vertex $v \in V(G)$ whose only two neighbors is u and w, we remove v and connect u and w with an edge. (Note that H now has only vertices with degree more than 2.)

1. Suppose we build a breadth-first tree starting from some vertex r of H. Prove that by the time we reach depth $\lg |V(H)|$, we will have discovered a cycle of length at most $2 \lg |V(H)|$.

2. Prove Erdős-Posá theorem.

This theorem suggests the following algorithm for FEEDBACK VERTEX SET (FVS).

FVS-1

1: $\mathbf{y} \leftarrow 0$
2: $C \leftarrow \emptyset$
3: Let G' be a graph obtained from G by removing all uninteresting vertices.
4: **while** G' is not empty **do**
5: Choose cycle k in G' of length at most $4 \lg |V(G')|$
6: // note that this cycle corresponds uniquely to a cycle in the original graph G
7: Increase y_k until there is some saturated vertex v
8: Add v into C
9: Remove v from G' and then all uninteresting vertices from G'
10: **end while**
11: Return C (call it $\bar{C}$)

The following theorem is now immediate from the above analysis.

Theorem 7.3. *Algorithm* FVS-1 *is a $4 \lg n$-approximation algorithm for* FEEDBACK VERTEX SET.

A 2-approximation for this problem can be obtained with a different integer programming formulation.

7.2 Refining the final solution

Consider the s-t SHORTEST PATH problem and how algorithm PRIMAL-DUAL BASIC applies to it. In this case, we want to pick a minimum-weight subset of edges which cover all s-t cuts $\delta(X)$.

Consider a typical iteration of the algorithm with the current set C of edges chosen so far. If s and t is not connected in (V, C), then there will be a number of "uncovered" cuts to choose from. A sensible approach is to choose a minimal cut that contains s, i.e. we chose $\delta(X)$ where X is the set of vertices of the connected component of (V, C) that contains s. It is not difficult to see that this strategy corresponds to Dijkstra's algorithm for the SINGLE SOURCE SHORTEST PATH problem.

Exercise 11. Prove that the strategy above corresponds to Dijkstra's algorithm.

Unfortunately, this algorithm produces redundant edges, which are in a shortest path tree rooted at s. Hence, it makes sense to remove redundant edges from the final solution $\bar{C}$. For some problems, it is better to remove redundant elements in the reverse order of their addition. This step is called the *reverse deletion step* and is illustrated in the following refinement of the basic algorithm.

PRIMAL-DUAL WITH REVERSE DELETION

1: $\mathbf{y} \leftarrow 0$
2: $C \leftarrow \emptyset$
3: $j \leftarrow 0$
4: **while** C is not a cover **do**
5: $\quad j \leftarrow j + 1$
6: $\quad k \leftarrow$ UNCOVERED-ELEMENT(C) // we can adapt this procedure for different problems
7: $\quad$ Increase y_k until $\exists S : \sum_{i \in S} y_i = w_S$
8: $\quad$ Refer to S as S_j and add it into C
9: **end while**
10: $\bar{C} \leftarrow$ REVERSE DELETE(C)

REVERSE DELETE(C)

1: **for** $j = |C|$ **downto** 1 **do**
2: $\quad$ **if** $C - \{S_j\}$ is feasible **then**
3: $\quad\quad$ remove S_j from C
4: $\quad$ **end if**
5: **end for**
6: Return C

Fix a k for which $y_k > 0$. As before we would like to estimate an upper bound for $g(\bar{C}, k)$. The situation becomes a little bit more complicated because of the reverse deletion. For notational conveniences, let $k(C) =$ UNCOVERED-ELEMENT(C), where we assume a deterministic strategy of choosing a k given a C.

At the point where we are about to increase y_k, the current solution is $C = \{S_1, \ldots, S_{j-1}\}$ for some j and $k = k(C)$ is not in any of these sets. Thus,

$$g(\bar{C}, k) = g\big(\bar{C} \cup C, k(C)\big).$$

The collection $A = \bar{C} \cup C$ is a *minimal augmentation* of C in the sense that removing any set from $A - C$ will result in an infeasible solution. This follows from the reverse deletion step. The following theorem follows readily.

Theorem 7.4. *If for any iteration of algorithm* PRIMAL-DUAL WITH REVERSE DELETION *with its infeasible solution C,*

$$\max_{A \text{ : min. aug. of } C} g(A, k(C)) \leq \beta$$

then the algorithm has approximation ratio β.

Exercise 12. Suppose we apply PRIMAL-DUAL WITH REVERSE DELETION to the s-t SHORTEST PATH problem, using the rule of picking a minimal cut containing s each time. Show that the $\beta = 1$ satisfies Theorem 7.4. In other words, the algorithm returns an optimal solution.

Exercise 13. In the MINIMUM-COST ARBORESCENCE problem, we are given a directed edge-weighted graph $G = (V, E)$ and a root vertex r. The goal is to find a minimum-cost tree rooted at r all of whose edges are directed away from r.

This problem can be viewed as a special case of WEIGHTED SET COVER in the following sense: for each subset X of $V - \{r\}$, we want the minimum-cost set of edges to cover $\delta^-(X)$, where $\delta^-(X) = \{(u, v) \mid u \notin X, v \in X\}$. (Here, (u, v) covers $\delta^-(X)$ iff $(u, v) \in \delta^-(X)$.)

Suppose we apply PRIMAL-DUAL WITH REVERSE DELETION to this problem.

(a) Consider an infeasible set of edges C in some iteration of the algorithm and the graph $G' = (V, C)$. Show that there is a strongly connected component of G' with vertex set X such that $r \notin X$, and $C \cap \delta^-(X) = \emptyset$.

(b) Describe how to find this component in polynomial time.

(c) Let UNCOVERED-ELEMENT(C) return $\delta^-(X)$ with X found as in (a). Apply Theorem 7.4 to show that this algorithm has approximation ratio 1, i.e. it returns an optimal solution.

7.3 Increasing simultaneously multiple dual variables

A MINIMUM SPANNING TREE (MST) instance can be view as a SET COVER instance where the edges need to cover all non-trivial cuts in the graph. Prim's algorithm for MST can be thought of as a special case of PRIMA-DUAL BASIC. However, Kruskal's algorithm is different. Kruskal's algorithm corresponds to increasing all dual variables simultaneously at the same rate until some edge becomes saturated. This idea is summarized in the following more general form of the primal dual method.

GENERAL PRIMAL-DUAL

1: $\mathbf{y} \leftarrow 0$
2: $C \leftarrow \emptyset$
3: $j \leftarrow 0$
4: **while** C is not a cover **do**
5: $\quad j \leftarrow j + 1$
6: $\quad \nu_j \leftarrow$ UNCOVERED-ELEMENTS(C) // pick a subset of uncovered elements
7: $\quad$ Increase all y_k at the same rate, $k \in \nu_j$ until $\exists S : \sum_{i \in S} y_i = w_S$
8: $\quad$ Refer to S as S_j and add it into C
9: **end while**
10: $\bar{C} \leftarrow$ REVERSE DELETE(C)

Let l be the total number of iterations. Let ϵ_j be the amount by which each variable in ν_j was increased. It is clear that, at the end we have

$$\sum_{k \in U} y_k = \sum_{j=1}^{l} |\nu_j| \epsilon_j.$$

Following our usual analysis, we have

$$\begin{aligned}\text{cost}(x^A) &= \sum_{i\in U} g(\bar{C}, i) y_i \\ &= \sum_{i\in U} g(\bar{C}, i) \sum_{j:\nu_j \ni i} \epsilon_j \\ &= \sum_{j=1}^{l} \left(\sum_{i\in\nu_j} g(\bar{C}, i) \right) \epsilon_j\end{aligned}$$

Let $\nu(C)$ denote UNCOVERED-ELEMENTS(C). The following theorem follows naturally.

Theorem 7.5. *If for any iteration j of algorithm* GENERAL PRIMAL-DUAL *with infeasible solution C,*

$$\max_{A\,:\,\text{min. aug. of } C} \sum_{i\in\nu(C)} g(A, i) \le \gamma |\nu(C)|$$

then the algorithm has approximation ratio γ.

We shall apply this algorithm to get a 2-approximation for the GENERALIZED STEINER TREE problem. Recall that we have an edge-weighted graph $G = (V, E)$ and m pairs of vertices (s_j, t_j) and we need to find a minimum-cost set of edges covering all s_j-t_j cuts. In this algorithm, $\nu(C) =$ UNCOVERED-ELEMENTS(C) is the set of all cuts $\delta(X)$ where X is a connected component of (V, C) for which $|X \cap \{s_j, t_j\}| = 1$ for some j.

Theorem 7.6. *The algorithm for* GENERALIZED STEINER TREE *as described above has approximation ratio* 2*.*

Proof. Consider an infeasible solution C. The graph (V, C) has several connected components. If A is a minimal augmentation of C, then the graph (V, A) is a forest if we view the connected components of (V, C) as vertices. Let T denote this forest.

The forest T has two types of vertices: the *red* vertices correspond to the connected components X where $\delta(X) \in \nu(C)$, and the rest are *blue* vertices. Let R denote the set of red vertices and B the set of blue vertices. We then have $|\nu(C)| = |R|$. Ignoring the blue vertices with degree 0, we have

$$\begin{aligned}\sum_{i\in\nu(C)} g(A, i) &= \sum_{v\in R} \deg_T(v) \\ &= 2|E(T)| - \sum_{v\in B} \deg_T(v) \\ &\le 2(|R| + |B|) - \sum_{v\in B} \deg_T(v) \\ &\le 2(|R| + |B|) - 2|B| \\ &= 2|\nu(C)|.\end{aligned}$$

The last inequality follows because no blue vertex has degree one, otherwise A is not a minimal augmentation of C. □

Exercise 14. Many of the problems we have discussed can be formulated with the following integer program. We assume that an edge-weighted graph $G = (V, E)$ is given, and edge e is weighted with $w_e \in \mathbb{Z}^+$.

$$\begin{aligned}\min \quad & \textstyle\sum_{e\in E} w_e x_e \\ \text{subject to} \quad & \sum_{e\in\delta(X)} x_e \ge f(X), \quad \emptyset \ne X \subset V, \\ & x_e \in \{0, 1\}, \qquad \forall e \in E.\end{aligned} \tag{20}$$

Here $\delta(X) = [X, \overline{X}]$, and $f : 2^V \to \mathbb{Z}^+$ is a function that counts how many edges must cross $\delta(X)$ in a feasible solution.

The dual of the LP relaxation of the above program is

$$\begin{aligned} \max \quad & \sum_{\emptyset \neq X \subset V} f(X) y_X \\ \text{subject to} \quad & \sum_{X : e \in \delta(X)} y_X \leq w_e, \qquad e \in E, \\ & y_X \geq 0, \quad \forall X, \emptyset \neq X \subset V. \end{aligned} \tag{21}$$

In this problem, we shall develop an approximation algorithm for this general setting where f is a special class of function. To solve this problem, you must understand thoroughly the algorithm we developed for the GENERALIZED STEINER TREE problem, which will be a special case of this problem.

We assume that f has the following properties:

- $f(X) \in \{0, 1\}$, for all $X \subseteq V$. In other words, f is a 01-function. This problem is thus a special case of our SET COVER problem in which each cut $\delta(X)$ with $f(X) = 1$ has to be covered.
- $f(V) = 0$. This is natural since $\delta(V) = \emptyset$.
- $f(X) = f(\overline{X})$ for all subsets X of V. This is also natural, since $\delta(X) = \delta(\overline{X})$ in an undirected graph.
- If X and Y are two disjoint subsets of V, then $f(X) = f(Y) = 0$ implies $f(X \cup Y) = 0$. This means if $\delta(X)$ and $\delta(Y)$ do not have to be covered, then so does $\delta(X \cup Y)$.

A function f satisfying the above properties is called a 01-proper function.

1. Let C be an infeasible subset of edges of G (with respect to f, of course). Prove that there is some connected component X of (V, C) for which $f(X) = 1$. (Here, we use X to also denote the set of vertices of the connected component X.)

2. Let C be an infeasible subset of edges of G. Let X be a connected component of (V, C). Let Y be a subset of vertices such that $Y \cap X \neq \emptyset$ and $X \not\subseteq Y$. Prove that C covers $\delta(Y)$.

   (Note: this means that we only have to worry about covering the $\delta(Y)$ for which Y contains one or a few connected components of (V, C).)

3. Consider the following algorithm for our problem.

   PRIMAL-DUAL FOR 01-PROPER FUNCTION

   1: $\mathbf{y} \leftarrow 0$; $C \leftarrow \emptyset$; $j \leftarrow 0$
   2: **while** C is infeasible **do**
   3: $\quad j \leftarrow j + 1$
   4: $\quad$ Let ν_j be the set of all X which is a connected component of (V, C) and $f(X) = 1$
   5: $\quad$ Increase all y_X at the same rate, $X \in \nu_j$, until $\exists e : \sum_{Z : e \in \delta(Z)} y_Z = w_e$
   6: $\quad$ Refer to e as e_j and add it into C
   7: **end while**
   8: $\overline{C} \leftarrow$ REVERSE DELETE(C)

   Prove that this is a 2-approximation algorithm for our problem.

Historical Notes

The primal-dual method was proposed by Dantzig, Ford, and Fulkerson [4] to solve linear programs. This method was motivated by the works of Egerváry [5] and Kuhn [15] on the so-called "Hungarian algorithm" for the assignment problem (or the minimum cost bipartite perfect matching problem). The primal-dual method is not effective as a method for solving linear programs in general. Its strength lies in the fact that it can be used to "transform" a weighted optimization problem into a purely combinatorial and unweighted optimization problem. Many fundamental combinatorial optimization algorithms are either a special case of this method or can be understood in terms of it. For more details, consult standard combinatorial optimization textbooks such as [3, 10, 16–19].

Bar-Yehuda and Even [2] gave the first truly primal-dual algorithm to approximate the VERTEX COVER and the SET COVER problem, as presented in algorithm WSC-PRIMAL-DUAL-A. The LP-algorithms for these problems were due to Hochbaum [12]. The algorithm for the GENERAL COVER problem is by Hall and Hochbaum [11]. Chapter 3 of [13] is a good survey on covering and packing problems.

The survey paper by Goemans and Williamson [9] and many chapters in Vazirani [20] discuss the primal-dual method for approximation in more details.

The book by Ahuja, Magnanti and Orlin [1] contains extensive discussions on network flows, related problems and applications.

The Max-Flow Min-Cut theorem was obtained independently by Elias, Feinstein, and Shannon [6], Ford and Fulkerson [8]. The special case with integral capacities was also discovered by Kotzig [14].

References

[1] R. K. AHUJA, T. L. MAGNANTI, AND J. B. ORLIN, *Network flows*, Prentice Hall Inc., Englewood Cliffs, NJ, 1993. Theory, algorithms, and applications.

[2] R. BAR-YEHUDA AND S. EVEN, *A linear-time approximation algorithm for the weighted vertex cover problem*, J. Algorithms, 2 (1981), pp. 198–203.

[3] W. J. COOK, W. H. CUNNINGHAM, W. R. PULLEYBLANK, AND A. SCHRIJVER, *Combinatorial optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Inc., New York, 1998. A Wiley-Interscience Publication.

[4] G. B. DANTZIG, L. R. FORD, JR., AND D. R. FULKERSON, *A primal-dual algorithm for linear programs*, in Linear inequalities and related systems, Annals of Mathematics Studies, no. 38, Princeton University Press, Princeton, N. J., 1956, pp. 171–181.

[5] J. EGERVÁRY, *Matrixok kombinatorikus tulajdonságairól*, Mathematikai és Fizikai Lápok, 38 (1931), pp. 19–28.

[6] P. ELIAS, A. FEINSTEIN, AND C. E. SHANNON, *Note on maximal flow through a network*, IRE Transactions on Information Theory IT-2, (1956), pp. 117–199.

[7] P. ERDŐS AND L. PÓSA, *On the maximal number of disjoint circuits of a graph*, Publ. Math. Debrecen, 9 (1962), pp. 3–12.

[8] L. R. FORD, JR. AND D. R. FULKERSON, *Maximal flow through a network*, Canad. J. Math., 8 (1956), pp. 399–404.

[9] M. X. GOEMANS AND D. WILLIAMSON, *The primal-dual method for approximation algorithms and its application to network design problems*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing Company, 1997, pp. 144–191.

[10] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *Geometric algorithms and combinatorial optimization*, vol. 2 of Algorithms and Combinatorics, Springer-Verlag, Berlin, second ed., 1993.

[11] N. G. HALL AND D. S. HOCHBAUM, *A fast approximation algorithm for the multicovering problem*, Discrete Appl. Math., 15 (1986), pp. 35–40.

[12] D. S. HOCHBAUM, *Approximation algorithms for the set covering and vertex cover problems*, SIAM J. Comput., 11 (1982), pp. 555–556.

[13] D. S. HOCHBAUM, ed., *Approximation Algorithms for NP Hard Problems*, PWS Publishing Company, Boston, MA, 1997.

[14] A. KOTZIG, *Súvislost' a pravideliná súvislost' konečných grafov*, Bratislava: Vysoká Škola Ekonomická, (1956).

[15] H. W. KUHN, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quart., 2 (1955), pp. 83–97.

[16] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1982.

[17] A. SCHRIJVER, *Combinatorial optimization. Polyhedra and efficiency. Vol. A*, vol. 24 of Algorithms and Combinatorics, Springer-Verlag, Berlin, 2003. Paths, flows, matchings, Chapters 1–38.

[18] ———, *Combinatorial optimization. Polyhedra and efficiency. Vol. B*, vol. 24 of Algorithms and Combinatorics, Springer-Verlag, Berlin, 2003. Matroids, trees, stable sets, Chapters 39–69.

[19] ———, *Combinatorial optimization. Polyhedra and efficiency. Vol. C*, vol. 24 of Algorithms and Combinatorics, Springer-Verlag, Berlin, 2003. Disjoint paths, hypergraphs, Chapters 70–83.

[20] V. V. VAZIRANI, *Approximation algorithms*, Springer-Verlag, Berlin, 2001.