APPLICATIONS OF LINEAR PROGRAMMING TO APPROXIMATION ALGORITHMS

JACK MANDELL

Abstract. Combinatorial optimization plays a vital role in areas such as operations research and computer science. When designing algorithms to solve combinatorial optimization problems, it is important to consider both their accuracy and efficiency at finding optima. However, many of the natural combinatorial optimization problems that arise are known to be NP-hard, so hope for polynomial-time algorithms is slim. By easing the requirement of finding true optimal solutions, approximation algorithms provide a framework for balancing optimally and runtime. In this paper, we explore how approximation algorithms can be created for various NP-hard problems by adapting techniques from linear programming.

1. Introduction

A combinatorial optimization problem can be described as

$$\begin{array}{ll} \text{minimize (or, maximize)} & c(x_1, x_2 \ldots, x_n) \\ \text{subject to} & (x_1, x_2 \ldots, x_n) \in \Omega \end{array}$$

where $\Omega \subseteq \mathbb{R}^n$ is called the **feasible region**, and $c : \Omega \to \mathbb{R}$ is an **objective function** to be optimized. We say that any $x \in \Omega$ is a **feasible solution** to the problem.

To motivate a framework for describing problems in combinatorial optimization, we will define a set of optimizations problems known as Minimum Weight Vertex Cover. We first define a concept in graph theory.

Definition 1.1. Let $G = (V, E)$ be a graph with $V = [n]$ and let $C \subseteq V$. C is a **vertex cover** if for any $\{i, j\} \in E$, either $i \in C$ or $j \in C$.

Example 1.2. Minimum Weight Vertex Cover (Min-WVC): Given a graph $G = (V, E)$ with non-negative weight function $c : V \to \mathbb{R}^+$, find a vertex cover of minimum total weight.

Note that MIN-WVC is not just a single combinatorial optimization problem, but really an *infinite set* of combinatorial optimization problems: there is a different combinatorial optimization problem for each different choice of graph G and cost function c. We would like to define a framework that captures this idea.

We say that a **class** of combinatorial optimization problems Π is a set of optimization problems that share a common structure and can be parameterized by a set of input values. We call any assignment of these input values an **instance**. If I is an instance of Π, we let $opt_\Pi(I)$ be the optimal value of Π, or $opt(I)$ if Π is understood. To demonstrate how the terminology is used in practice, we apply it to the class MIN-WVC.

In the above example, Π is MIN-WVC, and instance I is a graph $G = (V, E)$ together with a cost function c on the vertices. Ω is the set of all vertex covers, and $opt_\Pi(I)$ is the weight of the optimal vertex cover in instance I.

If A is an algorithm to solve a class of optimization problems Π, and $A(I)$ is the output value produced by A in instance I, then ideally, we would like that $opt(I) = A(I)$. Instead, for an approximation algorithm, we allow $A(I)$ to be "close" to $opt(I)$. To quantify how exact the algorithm is, we use the concept of a performance ratio. This will give an measure of the worst-case inaccuracy.

Definition 1.3. The **performance ratio** of an approximation algorithm A for optimization problem Π is defined as

$$r(A) := \sup_I \frac{A(I)}{opt(I)}$$

if Π is a class of minimization problems, and

$$r(A) := \inf_I \frac{opt(I)}{A(I)}$$

if Π is a class of maximization problems.

Note that the performance ratio gives a bound on the worst case performance of A, which is typically of most concern to computer scientists. While this definition captures the algorithm's performance quite nicely, it is often too difficult to show some number is indeed the performance ratio. This involves constructing instances I such that the ratio of $A(I)$ to $opt(I)$ is arbitrarily close to $r(A)$. However, the entire premise of approximation algorithm theory assumed that $opt(I)$ is too hard to calculate, and so it would then be impossible to calculate the ratio. The performance ratio, however, motivates the following

weaker definition which is used in practice to analyze the performance of approximation algorithms.

Definition 1.4. Let $k \geq 1$. A algorithm A is a ***k*-factor approximation** for Π if for any instance I,

$$A(I) \leq k \cdot opt(I) \tag{1.1}$$

if Π is a class of minimization problems, and

$$opt(I) \leq k \cdot A(I) \tag{1.2}$$

if Π is a class of maximization problems.

Consider the minimization form first of the k-factor approximation. If we divide by $opt(I)$ on both sides of (1.1), then we obtain

$$\frac{A(I)}{opt(I)} \leq k$$

This happens to look very similar to the definition of the performance ratio. Because we are only looking for an upper bound (as opposed to the least upper bound) of the ratios of $A(I)/opt(I)$, calculating $opt(I)$ is not needed, and this is exactly why this definition is more practical. In this paper, we will focus on finding approximation algorithms for minimization problems, so finding k that satisfies (1.1) will be our main goal.

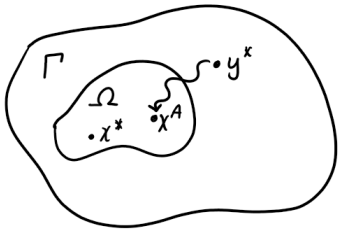
So far, we have defined accuracy of an approximation, and now we will need to discuss runtime. The notion of runtime of an approximation algorithm depends on the computational model at which the algorithm is defined. At a low level, there is with a Turing Machine, and at the high level, there is in terms of pseudocode. To truly define runtime, one must utilize the framework of Turing Machines, which can be used to define time complexity classes of P and NP. For the purpose of this paper, we will not resort to this level of detail. The level of pseudocode will suffice. If an algorithm is written as pseudocode, we define the runtime as the number of statements, such as logical comparisons or binary operations, that make up the algorithm. For an approximation algorithm A, we define the worst-case runtime on instance size n as the maximum runtime over all instances I with size n. This function of n will denote the **time complexity** of A. Lastly, an approximation algorithm runs in polynomial time if the time complexity is a polynomial in n.

Given an optimization problem, how does one go about creating k-factor approximation algorithms? The general technique that we will explore in this paper is as follows: for a combinatorial optimization problem, it is first posed as an integer linear program. This framework

allows one to define a cost function c and a set of linear inequality constraints that will define Ω. For the majority of the paper, we will use combinatorial optimization problems that arise in graph theory, which can be converted into integer linear programs quite naturally. Once the problem is converted into one of these programs, we can take advantage of the rich theory and methods from integer and linear programming to create an approximation algorithm. Two of these methods we will explore are **relaxation** and the **primal-dual scheme**.

Relaxation involves increasing the size of the feasible region Ω to some $\Gamma \supseteq \Omega$, where finding the optimum in Γ can be found in polynomial time. The optimal solution in Γ can then be converted, usually through some sort of rounding technique, into a feasible solution in Ω, which will be an approximate solution to the original optimization problem. To ensure the approximation algorithm runs in polynomial time, it is important to make sure that rounding can also be done in polynomial time. To analyze an approximation algorithms performance, the key will be estimating the loss in optimality that is generated when converting the optimum in Γ to a feasible solution in Ω.

To show how a k-factor approximation can be obtained, we will use the following analysis. Suppose an instance I of Π is the problem $c(x^*) = \min_{x\in\Omega} c(x)$. Now we relax the problem to $\Gamma \supseteq \Omega$ and let I' denote the instance of the problem that gives $c(y^*) = \min_{y\in\Gamma} c(y)$. Lastly, we round y^* to some $x^A \in \Omega$. We summarize the setup with the figure below:



Then since $\Omega \subseteq \Gamma$ and we are minimizing over both sets, it follows that $c(y^*) \leq c(x^*)$. Hence

$$\frac{c(x^A)}{c(x^*)} \leq \frac{c(x^A)}{c(y^*)}$$

To use the notation from the beginning of the introduction, we have that $A(I) = c(x^A)$, $opt(I) = c(x^*)$, and $opt(I') = c(y^*)$ and so

$$\frac{A(I)}{opt(I)} \leq \frac{A(I)}{opt(I')} \tag{1.3}$$

Since we are able to obtain $opt(I')$ in polynomial time, if we find a k such that

$$\frac{A(I)}{opt(I')} \leq k,$$

for any I, then by (1.3), we have that $A(I) \leq k \cdot opt(I)$ and so A is a k-factor approximation for the class of problems Π. We can repeat the above analysis for the maximization problem as well and obtain a similar conclusion.

2. Linear Programming

2.1. **Preliminaries.** A nice class of optimization problems are ones where the cost function c is linear, and Ω is the intersection of linear half-spaces. These type of optimization problems can be formulated into what are known as **Linear Programs** (LP). If only integer values in Ω are allowed, then it is called an **Integer Linear Program** (ILP). We will see that Min-WVC as well as several other combinatorial optimization problems can be posed as an ILP, which will form the basis for constructing approximation algorithms. Any linear program can be written in standard form as

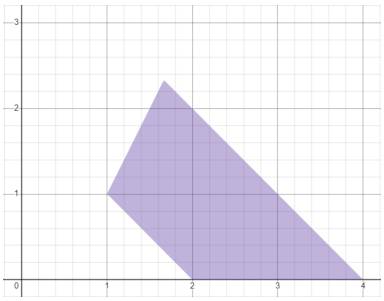
$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \geq b \\ & x \geq 0 \end{array} \tag{2.1}$$

where A is an $m \times n$ matrix with entries in $\mathbb{R}$, and both c and x are n-dimensional column vectors.

Example 2.1. The following is an example of a linear program

$$\begin{array}{ll} \text{minimize} & 3x_1 + x_2 \\ \text{subject to} & 2x_1 + 2x_2 \geq 4 \\ & -x_1 - x_2 \geq -4 \\ & 2x_1 - x_2 \geq 1 \\ & x_1, x_2 \geq 0 \end{array}$$

In this example, we see that the feasible region is a convex polygon in $\mathbb{R}^2$, and in general the feasible region need not be bounded.



We will now transform Min-WVC into an ILP. Let $G = (V, E)$ and $V = [n]$. If $C \subseteq V$ is the vertex cover with minimum weight, define

$$x_i := \begin{cases} 1 & \text{if } i \in C \\ 0 & \text{if } i \notin C \end{cases}$$

Since C is a vertex cover, for any edge $\{i, j\} \in E$, we must have $i \in C$ or $j \in C$ and so $x_i + x_j \geq 1$. Since $c(i)$ is the cost of including vertex i in the vertex cover, the ILP for Min-WVC can be stated as follows.

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{i=1}^{n} c(i)x_i & \\ \text{subject to} & x_i + x_j \geq 1 & \{i, j\} \in E \\ & x_i \in \{0, 1\} & i \in V \end{array} \tag{2.2}$$

Any LP can be solved in polynomial time. On the other hand, solving an ILP is NP-hard.

2.2. **Relaxation and Rounding.**

To find approximation algorithms for problems that can be stated as an ILP, the process of relaxation and rounding can be applied. An ILP is first converted into an LP by *relaxing* integrality conditions. Once this LP is solved, the solution to the linear program is *rounded* back into an approximate solution to the original ILP. We will see that there are various rounding techniques that can be employed.

2.2.1. *Threshold Rounding.*

For optimization problems that can be stated as binary integer linear programs, such as Min-WVC, the condition of $x_i \in \{0, 1\}$ can be relaxed to $0 \leq x_i \leq 1$. We now have the following linear program:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{i=1}^{n} c(i)x_i & \\ \text{subject to} & x_i + x_j \geq 1 & \{i,j\} \in E \\ & 0 \leq x_i \leq 1 & i \in V \end{array} \tag{2.3}$$

Then, if $x^* = (x_1^*, x_2^* \ldots, x_n^*)$ is the solution to (2.3), an approximate solution to (2.2) can be found by rounding each x_i^* to the nearest integer, essentially using the threshold 1/2. The algorithm can be stated as follows.

Algorithm 1 Threshold Rounding Approximation for MIN-WVC

1. Convert instance I of MIN-WVC into the integer linear program (2.2)
2. Relax the constraints of (2.2) to form the linear program (2.3)
3. For $i \in V$, let
$$x_i^A := \begin{cases} 1 & \text{if } x_i^* \geq 0.5 \\ 0 & \text{if } x_i^* < 0.5 \end{cases}$$
4. Output $x^A = (x_1^A, x_2^A \ldots, x_n^A)$

Theorem 2.2. *Algorithm 1 is a 2-factor approximation algorithm for* MIN-WVC.

Proof. First we show that x^A is a feasible solution to (2.2). For any $\{i,j\} \in E$, we have that $x_i^* + x_j^* \geq 1$. So, either $x_i^* \geq 0.5$ or $x_j^* \geq 0.5$. Hence, either $x_i^A = 1$ or $x_j^A = 1$, and so the constraints are satisfied. By step 3 in Algorithm 1, it follows that $x_i^A \leq 2x_i^*$. Hence

$$A(I) = \sum_{i=1}^{n} c(i)x_i^A \leq 2\sum_{i=1}^{n} c(i)x_i^* \tag{2.4}$$

If $opt_\Pi(I)$ is the optimal solution of (2.2) and $opt_\Pi(I')$ is the optimal solution to (2.3), by (2.4),

$$\frac{A(I)}{opt_\Pi(I)} \leq \frac{A(I)}{opt_\Pi(I')} \leq 2 \tag{2.5}$$

Hence, Algorithm 1 is a 2-factor approximation algorithm.

□

A consequence of Theorem (2.2) is that $r(A) \leq 2$. We can actually show a much stronger conclusion for Algorithm 1.

Theorem 2.3. *Algorithm 1 has performance ratio 2.*

Proof. To do this, we will construct a sequence of instances I_n, where $A(I_n)/opt(I_n)$ becomes arbitrarily close to 2.

Let C_n denote the cyclic graph of n vertices for $n \in \mathbb{N}$. One can think of this as an n-gon. Then take instance I_k to be C_{2k+1} together with the weight function $c(i) = 1$ for every $i \in V$. Since the weight of each vertex is the same and the graph is cyclic, it follows that an optimal vertex cover is $C = \{1, 3, 5, \ldots, 2k+1\}$, all odd vertices. Hence $opt(I_k) = k+1$. But, if we convert the instance I_k into the integer linear program (2.2) and solve the relaxation (2.3), we obtain that $x_i^* = \frac{1}{2}$ for every $i \in V$. Hence Algorithm 1 rounds all x_i to 1, and so $A(I_k) = 2k+1$. Therefore,

$$\frac{A(I_k)}{opt(I_k)} = \frac{2k+1}{k+1}$$

If we let $k \to \infty$, it follows that $r(A) \geq 2$. So, since $r(A) \leq 2$ as previously said, it follows that $r(A) = 2$.

□

Are there approximation algorithms for Min-WVC that have a better performance ratio than 2? In other words, a $(2-\epsilon)$-factor approximation for some $\epsilon \in (0, 1)$. In fact, if the Unique Games Conjecture is true, which relates to the approximability of various problems in Computer Science, then there are no polynomial-time algorithms with performance ratio better than 2! So, not only is it hard to find exact algorithms to solve Min-WVC in polynomial time, it is hard to find good *approximation* algorithms for Min-WVC in polynomial time!

2.2.2. *Iterated Rounding.*

One issue that may present when implementing threshold rounding for some integer linear program is that rounding down a combination of x_i^* may lead to a violation of one or more constraints. To address this issue, rounding can be performed in iterations to ensure that constraints are never violated. We will introduce the Minimum Generalized Spanning Network Problem to illustrate this method. First, we introduce some definitions and lemma from graph theory which will be helpful for the problem definition and construction of the integer linear program.

Definition 2.4. Given a graph $G = (V, E)$ and $k \in \mathbb{Z}$. G is ***k*-edge-connected** if $G' = (V, E \setminus F)$ is connected for any $F \subseteq E$ with $|F| < k$.

Definition 2.5. Let $G = (V, E)$. A **cut** of G, denoted by $(S, V \setminus S)$ for some $\emptyset \neq S \subset V$, is a partition of V. The **cut-set** of a cut $(S, V \setminus S)$, denoted by $\delta_G(S)$, are the set of edges in G with one vertex in each S and $V \setminus S$.

Example 2.6. Generalized Spanning Network (Min-GSN): Given a graph $G = (V, E)$ with non-negative cost function $c : E \to \mathbb{Z}^+$ on edges and $k \in \mathbb{Z}$, find a k-edge-connected subgraph of minimum weight.

Lemma 2.7. *G is k-edge-connected if and only if $|\delta_G(S)| \geq k$ for any cut $(S, V \setminus S)$.*

Proof. Assume that there exists a cut $(S, V \setminus S)$ such that $|\delta_G(S)| = l < k$. Then deleting these l edges from F makes the subgraph G' disconnected, which means that G' can not be k-edge-connected.

Suppose that G is not k-edge-connected. Then there exists some $F \subseteq E$ with $|F| < k$ such that $G' = (V, E \setminus F)$ is disconnected. Let $(H_1, \ldots, H_m)$ be the connected components of G'. Let $F_1 \subseteq F$ denote the set of edges with one vertex in H_1 and one in H_i for $i \neq 1$. Then $F_1 = \delta_G(S) \subseteq F$ where S is the set of vertices in component H_1. We found a cut $(S, V \setminus S)$ with $|\delta_G(S)| < k$.

□

We first convert Min-GSN into an integer linear program. Let $G' = (V, F)$ denote the optimal subgraph, where $F \subseteq E$. Note that G' can be determined solely by knowing F. Hence, to find the optimal subgraph we need only determine F. Denote

$$x_e := \begin{cases} 1 & \text{if } e \in F \\ 0 & \text{if } e \notin F \end{cases}$$

By lemma (2.7), G' is k-edge-connected if and only if $|\delta_{G'}(S)| \geq k$ for any cut $(S, V \setminus S)$. But, $|\delta_{G'}(S)| = \sum_{e \in \delta_G(S)} x_e$. Hence, we can represent Min-GSN in the following integer linear program:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c(e) x_e \\ \text{subject to} \quad & \sum_{e \in \delta_G(S)} x_e \geq k \quad \emptyset \neq S \subsetneq V \\ & x_e \in \{0, 1\} \end{aligned} \tag{2.6}$$

We can relax (2.6) to the following linear program:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c(e) x_e \\ \text{subject to} \quad & \sum_{e \in \delta_G(S)} x_e \geq k \quad \emptyset \neq S \subsetneq V \\ & 0 \leq x_e \leq 1 \qquad e \in E \end{aligned} \tag{2.7}$$

Definition 2.8. Let $f : 2^V \to \mathbb{Z}$. We say that f is *weakly supmodular*, if $f(V) = 0$ and for any $A, B \subseteq V$

$$f(A) + f(B) \leq f(A \setminus B) + f(B \setminus A)$$

or

$$f(A) + f(B) \leq f(A \cup B) + f(B \cap A)$$

We have the following lemma, which will be crucial to the iterated rounding approximation algorithm.

Lemma 2.9. *Let f be weakly supmodular, then any feasible solution x^* to*

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c(e) x_e \\ \text{subject to} \quad & \sum_{e \in \delta_G(S)} x_e \geq f(S) \quad \emptyset \neq S \subset V \\ & 0 \leq x_e \leq 1 \qquad\qquad e \in E \end{aligned} \tag{2.8}$$

has a component $x^*_e \geq 1/3$.

We can transform (2.7) into the form (2.8) with f defined as follows:

$$f(S) := \begin{cases} 0 & \text{if } S = \emptyset, V \\ k & \text{otherwise} \end{cases} \tag{2.9}$$

f is a weakly supmodular function. Suppose that we round the $x_e \geq 1/3$ to 1. Then, these edges are now part of the edge set F. We now write a new linear but for the graph with the edges that have been added to F deleted from the graph G.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E \setminus F} c(e) x_e \\ \text{subject to} \quad & \sum_{e \in \delta_{(V, E \setminus F)}(S)} x_e \geq f(S) - |\delta_{(V, E \setminus F)}(S)| \quad \emptyset \neq S \subsetneq V \\ & 0 \leq x_e \leq 1 \qquad\qquad e \in E \end{aligned} \tag{2.10}$$

Then $f(S) - |\delta_{(V,E \setminus F)}(S)|$ is again weakly supmodular, and so Lemma (2.9) applies. We can re-solve this program and repeat the rounding step. By formalizing this process into pseudocode, we have the following approximation algorithm for Min-GSN.

Algorithm 2 Iterated Rounding Approximation for MIN-GSN

1. Transform MIN-GSN into the integer linear program (2.6).
2. Relax the constraints of (2.6) to form the linear program (2.7)
3. Let f be defined as in (2.9), and let $F = \emptyset$
4. While F is not k-edge-connected
5. Compute optimal solution x^* of (2.10)
6. $F = F \cup \{x_e^* \in x^* | x_e^* \geq 1/3\}$
7. Output F

Theorem 2.10. *Algorithm 2 is a* 3*-factor approximation algorithm for* MIN-GSN*.*

Proof. Let F_i denote the edges e added to F after the first i iterations. Also, let $\overline{F}_i = E \setminus F_i$. Suppose that the maximum number of iterations completed is t and so in particular $F_t = F$. Denote x^0 as the solution to (2.10) with $F = \emptyset$ and denote x^i as the solution to (2.10) with $F = F_i$ for $i \in [t]$. Then since we are deleting edges from the graph each iteration and resolving the linear program (2.10), it follows that $x_e^{i-1} \leq x_e^i$ for any $i \in [t]$ and $e \in E$. Then, since $x_e = 1$ if $e \in F_i$, we have that

$$A(I) = \sum_{e \in F} c(e) = \sum_{e \in F_{t-1}} c(e) + \sum_{e \in \overline{F}_{t-1}} c(e)$$

Then for any $e \in \overline{F}_{t-1}$, it follows that $1 \leq 3x_e^{t-1}$. Hence:

$$A(I) \leq \sum_{e \in F_{t-1}} c(e) + 3 \sum_{e \in \overline{F}_{t-1}} c(e) x_e^{t-1}$$

Since $x_e^{t-1} \leq x_e^{t-2}$

$$A(I) \leq \sum_{e \in F_{t-1}} c(e) + 3 \sum_{e \in \overline{F}_{t-1}} c(e) x_e^{t-2}$$

If $e \in F_{t-1}$, but $e \notin F_{t-2}$, then e was added at iteration $t-1$. So, $3x_e^{t-2} \geq 1$. So

$$A(I) \leq \sum_{e \in F_{t-2}} c(e) + 3 \sum_{e \in \overline{F}_{t-2}} c(e) x_e^{t-2}$$

Now we repeat this process again but with iteration $i = t-3$ to obtain

$$\begin{aligned} A(I) &\leq \sum_{e \in F_{t-2}} c(e) + 3 \sum_{e \in \overline{F}_{t-2}} c(e) x_e^{t-3} \\ &\leq \sum_{e \in F_{t-3}} c(e) + 3 \sum_{e \in \overline{F}_{t-3}} c(e) x_e^{t-3} \end{aligned}$$

By, repeating until going all the way down to $i = 0$,

$$\begin{aligned} &\leq \sum_{e \in F_0} c(e) + 3 \sum_{e \in \overline{F}_t} c(e) x_e^0 \\ &= 3 \sum_{e \in E} c(e) x_e^0 \leq 3 \cdot opt(I) \end{aligned}$$

Hence Algorithm 2 is a 3-factor approximation. □

2.2.3. *Random Rounding.*
Round fractional values randomly to an integer. We can obtain pretty good expected performances, and the algorithms can be derandomized in practice using conditional expectation.

Definition 2.11. Given a graph $G = (V, E)$, $F \subseteq E$ is an **edge cut** if $G' = (V, E \setminus F)$ has two connected components.

Example 2.12. Minimum Feasible Cut (Min-FC):
Given a graph $G = (V, E)$ with edge weight $c : E \to \mathbb{R}^+$, a vertex $s \in V$, and a set M of pairs of vertices in G, find a subset of V with the minimum-weight edge cut that contains s but does not contain any pair in M.

We will first transform Min-FC into an integer linear program. Let S denote the optimal subset of vertices and F be the minimum-weight edge cut. Let

$$y_i := \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{if } i \notin S \end{cases}$$

$$x_e := \begin{cases} 1 & \text{if } e \in F \\ 0 & \text{if } e \notin F \end{cases}$$

To ensure that S does not contain any pair of vertices in M, for any $\{i, j\} \in M$, we must have that $y_i + y_j \leq 1$. To ensure that $s \in V$, we need $y_s = 1$. Lastly, to make sure that F is in fact the set of edges that corresponds to the edge-cut generated by S, we must have the following conditions: $x_e = 1$ if and only if $(y_i, y_j) = (1, 0)$ or $(y_i, y_j) = (0, 1)$ and

$x_e = 0$ if and only if $(y_i, y_j) = (0,0)$ or $(y_i, y_j) = (1,1)$. We can simplify this condition to $x_e \geq |y_i - y_j|$. This inequality can then be decomposed into two constraints, $x_e \geq y_i - y_j$ and $x_e \geq y_j - y_i$, to avoid using absolute values in the linear inequality. In summary, we can represent MIN-FC in the following integer linear program:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{e \in E} c(e) x_e & \\ \text{subject to} & x_e \geq y_i - y_j & e = \{i,j\} \in E \\ & x_e \geq y_j - y_i & e = \{i,j\} \in E \\ & y_i + y_j \leq 1 & \{i,j\} \in M \\ & y_s = 1 & \\ & y_i, x_e \in \{0,1\} & i \in V, e \in E \end{array} \tag{2.11}$$

We can relax (2.11) to the following linear program:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{e \in E} c(e) x_e & \\ \text{subject to} & x_e \geq y_i - y_j & e = \{i,j\} \in E \\ & x_e \geq y_j - y_i & e = \{i,j\} \in E \\ & y_i + y_j \leq 1 & \{i,j\} \in M \\ & y_s = 1 & \\ & 0 \leq y_i, x_e \leq 1 & i \in V, e \in E \end{array} \tag{2.12}$$

We can create an approximation algorithm for MIN-FC as follows:

Algorithm 3 Random Rounding Approximation for MIN-FC

1. Convert instance I of MIN-FC into the integer linear program (2.11)
2. Relax the constraints of (2.11) to the instance I' to form the linear program (2.12)
3. Generate $U \sim \text{Unif}(1/2, 1)$ and if (x^*, y^*) is the optimal solution to (2.12), then for $i \in V$ let
   $y_i^A := \begin{cases} 1 & \text{if } y_i^* \geq U \\ 0 & \text{if } y_i^* < U \end{cases}$
4. For $e = \{i,j\} \in E$, let $x_e^A = |y_i^A - y_j^A|$
5. Output (x^A, y^A)

Theorem 2.13. *Algorithm 3 is a 4-factor approximation algorithm for* MIN-FC.

Proof. First we show that (x^A, y^A) is a feasible solution. Since $U \geq \frac{1}{2}$, for any edge $e = \{i,j\}$, either $y_i \geq U$ or $y_j \geq U$, but not both. Hence, only one of y_i^A or y_j^A is equal to 1 and so $y_i + y_j \leq 1$. Also, since

$x_e^A := |y_i^A - y_j^A|$, it follows that both $x_e^A \geq y_j^A - y_i^A$ and $x_e^A \geq y_i - y_j$. Now, we show the performance bound. By linearity of expectation,

$$E\left[\sum_{e\in E} c(e)x_e^A\right] = \sum_{e\in E} c(e)\cdot E[x_e^A]$$

Fix some $e = \{i,j\} \in E$. Since $x_e^A = |y_i^A - y_j^A|$ and $y_i^A, y_j^A \in \{0,1\}$, it follows that $x_e^A \in \{0,1\}$ and so $E[x_e^A] = P(x_e^A = 1)$. Since there are two possible combinations of y_i^A and y_j^A that yield $x_e^A = 1$,

$$P(x_e^A = 1) = P(y_i^A = 1, y_j^A = 0) + P(y_i^A = 0, y_j^A = 1)$$

We will find a bound on $P(y_i^A = 1, y_j^A = 0)$. By construction, $y_i^A = 1$ only if $y_i^* \geq U$ and similarly $y_j^A = 0$ only if $y_j^* < U$. Hence,

$$P(y_i^A = 1, y_j^A = 0) = P(U \in (y_j^*, y_i^*))$$

Since U is drawn uniformly at random from $[\frac{1}{2}, 1]$ and $y_i^A - y_j^A \leq x_e^*$, it follows that

$$P(y_i^A = 1, y_j^A = 0) \leq 2x_e^*$$

We can repeat the above argument with slight modification to show

$$P(y_i^A = 0, y_j^A = 1) \leq 2x_e^*$$

as well. Hence, $E[x_e^A] \leq 4x_e^*$ and

$$E\left[\sum_{e\in E} c(e)x_e^A\right] \leq 4\sum_{e\in E} c(e)x_e^* = 4\cdot opt_\Pi(I')$$

$$\frac{E\left[\sum_{e\in E} c(e)x_e^A\right]}{opt_\Pi(I)} \leq \frac{E\left[\sum_{e\in E} c(e)x_e^A\right]}{opt_\Pi(I')} \leq 4$$

□

2.3. **Primal-Dual Schema.**

A rich theory in linear programming is duality theory. Given a linear program, known as the primal problem, one can construct a complement to it, known as the dual linear program. We will see that the primal and dual linear programs will have a special relationship that will be useful for constructing approximation algorithms. To motivate the dual linear program, we will use the example from earlier in the

paper. Recall Example (2.1):

$$\begin{array}{ll} \text{minimize} & 3x_1 + x_2 \\ \text{subject to} & 2x_1 + 2x_2 \geq 4 \\ & -x_1 - x_2 \geq -4 \\ & 2x_1 - x_2 \geq 1 \\ & x_1, x_2 \geq 0 \end{array} \tag{2.13}$$

One can show that $x^* = (1, 1)$ and $3x_1^* + x_2^* = 4$ is the minimum value. However, instead of thinking about minimizing the objective function, what if we instead try to maximize lower bounds of the objective function implied by linear combinations of the constraints? To find lower bounds on the objective function, we use the fact that any feasible solution not only satisfies the constraints, but satisfies *linear combinations* of the constraints with positive coefficients. However, only certain types of these linear combinations generate a lower bound. We give an example to demonstrate. Let I_i denote i^{th} inequality of the linear program (2.13) for $i \in \{1, 2, 3\}$. Since x^* is a feasible solution, it must satisfy I_i for all i and in particular $I_1 + \frac{1}{2}I_2$. Hence,

$$(2x_1 + 2x_2) + \frac{1}{2}(-x_1 - x_2) \geq 4 + \frac{1}{2}(-4)$$

Therefore $\frac{3}{2}x_1 + x_2 \geq 2$. Since the the x_i are non-negative, we have

$$3x_1 + x_2 \geq \frac{3}{2}x_1 + x_2 \geq 2$$

This means the minimum value must be at least 2. This linear combination gave us a lower bound on the optimal value because of two reasons. Firstly, $I_1 + \frac{1}{2}I_2$ is a linear combination with non-negative coefficients. This ensured the signs on the inequalities were preserved. Secondly, the coefficient in front of each x_i in $I_1 + \frac{1}{2}I_2$ was less than or equal to the coefficient of x_i in the objective function, for $i = 1, 2$. Hence, if $y_1I_1 + y_2I_2 + y_3I_3$ is to give a lower bound on the objective function of (2.13), the y_i must satsify

$$\begin{array}{l} 2y_1 - y_2 + 2y_3 \leq 3 \\ 2y_1 - y_2 - y_3 \leq 1 \\ y_1, y_2, y_3 \geq 0 \end{array}$$

To maximize the lower bound, we then generate the linear program

$$\begin{array}{ll} \text{maximize} & 4y_1 - 4y_2 + y_3 \\ \text{subject to} & 2y_1 - y_2 + 2y_3 \leq 3 \\ & 2y_1 - y_2 - y_3 \leq 1 \\ & y_1, y_2, y_3 \geq 0 \end{array} \tag{2.14}$$

Linear program (2.14) is known as the *dual* of linear program (2.13). If we recall the standard form primal

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \geq b \\ & x \geq 0 \end{array} \tag{2.15}$$

then the dual linear program is

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y \leq c \\ & y \geq 0 \end{array} \tag{2.16}$$

Theorem 2.14 (Weak Duality Theorem)**.** *If x and y are feasible solutions to linear programs (2.15) and (2.16), respectively, then we have that $c^T x \geq b^T y$.*

Proof. Since x and y are feasible, we have that

$$c^T x \geq (A^T y)^T x = y^T A x \geq y^T b = (y^T b)^T = b^T y \tag{2.17}$$

□

Theorem 2.15 (Strong Duality Theorem)**.** *The primal problem has a finite optimum iff its dual has a finite optimum. The optimal values are the same.*

The proof of the above theorem is not hard, but is tedious. It involves the use of Gaussian Elimination to eliminate variables from the linear program. We omit this from the paper.

We can use Weak and Strong Duality to derive the complementary slackness conditions, which test whether two given feasible solutions x and y of the primal and dual linear program are optimal. These conditions will play a major role in the primal-dual scheme.

Definition 2.16. If x and y are feasible solutions to linear programs (2.15) and (2.16), we say that x and y satisfy the **complementary slackness conditions** if

(1) $x_j > 0 \implies (A^T y)_j = c_j$
(2) $y_i > 0 \implies (Ax)_i = b_i$

The conditions (1) and (2) are also known as the primal and dual complementary slackness conditions, respectively. We say that a constraint is **tight** if equality holds.

Lemma 2.17. *x and y are feasible solutions to linear programs (2.15) and (2.16) that satisfy the complementary slackness conditions if and only if x and y are optimal solutions.*

Proof. For any component x_j of x, we must have by complimentary slackness that $x_j(c - A^T y)_j = 0$. Since if $x_j = 0$, then done, and if $x_j > 0$, then $(c - A^T y)_j = 0$ is assumed. Hence,

$$\sum_{j=1}^{n} x_j(c - A^T y)_j = 0$$

The above is equivalent to $x^T(c - A^T y) = 0$. Therefore, $c^T x = (A^T y)^T x$ and the first inequality in the proof of the Weak Duality Theorem becomes equality. We can show by a similar argument using the dual complementary slackness conditions that $y^T Ax = y^T b$. So, the last inequality of the proof of Weak Duality becomes equality, and so $c^T x = b^T y$. By Strong Duality, x and y are hence optimal solutions.

For the opposite direction, since x and y are optimal and hence feasible, $c^T x = b^T y$, and so by the proof of Weak Duality, we immediately obtain $c^T x = (A^T y)^T x$ and $y^T Ax = y^T b$. We will just show that the primal complimentary slackness condition follow from here, as the dual complementary slackness conditions are similar in argument. $c^T x = (A^T y)^T x$ is equivalent to $x^T(c - A^T y) = 0$. Since this inner product of vectors with non-negative entries equals zero, it must be the case that $x_j(c - A^T y)_j = 0$ for all $j \in [n]$. Therefore, if $x_j > 0$, then $(c - A^T y)_j = 0$.

□

The general idea behind the primal-dual scheme is to first start out with some infeasible solution x to the primal problem and some feasible solution y to the dual problem. We then will iteratively improve the dual solution y and adjust the x solution in response until it becomes feasible. This feasible x solution after iterating will be the approximate solution to the optimization problem. To formally define the primal-dual scheme, we will use the Minimum Hitting Set Problem.

Example 2.18. Minimum Hitting Set (Min-HS):
Given subsets $T_1, T_2, \ldots, T_p$ of a ground set E and a non-negative weight function $c : E \to \mathbb{Q}^+$, find a set $A \subseteq E$, such that $A \cap T_i \neq \emptyset$ for all $i \in [p]$ and the cost $\sum_{e \in A} c(e)$ is minimized.

Due to the structure of the above problem, many optimization problems can be easily posed as an instance of Min-HS. By writing the primal-dual scheme in this general case, we need only to write an optimization problem as an instance of Min-HS, and the primal-dual

approximation algorithm will follow. We will demonstrate this idea later on by converting the problem Minimum Feedback Vertex Set into an instance of MIN-HS. Since performance analysis will be done for the general problem, it will be easy to analyze the performance of the algorithm for any instance. As usual, denote x_e by

$$x_e := \begin{cases} 1 & \text{if } e \in A \\ 0 & \text{if } e \notin A \end{cases}$$

and so the integer linear program for MIN-FVS can be written as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c(e) x_e \\ \text{subject to} \quad & \sum_{e \in T_i} x_e \geq 1 \quad \forall i \in [p] \\ & x_e \in \{0, 1\} \end{aligned} \tag{2.18}$$

The dual of the relaxation of the integer linear program (2.18) can thus be written as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{p} y_i \\ \text{subject to} \quad & \sum_{i : e \in T_i} y_i \leq c(e) \quad \forall e \in E \\ & y_i \geq 0 \end{aligned} \tag{2.19}$$

We now motivate the primal-dual scheme algorithm. By applying the definitions, the primal and dual complementary slackness conditions are

$$\begin{aligned} x_e > 0 &\implies \sum_{i : e \in T_i} y_i = c_e \\ y_i > 0 &\implies \sum_{e \in T_i} x_e = 1 \end{aligned}$$

Suppose we start out with $x = (0, 0, \ldots, 0)$, an infeasible solution to (2.18), and also start with $y = (0, 0, \ldots, 0)$. Then there must exist some violated constraint in primal. More specifically, there is some $k \in [p]$ such that $\sum_{e \in T_k} x_e = 0$ (Currently, this holds for any $k \in [p]$). By definition, $y_k = 0$ in the dual program. If y_k is increased, then the dual solution y improves as the dual is a maximization problem. So, after increasing y_k just enough, there will be some $e \in T_k$, such that $\sum_{i : e \in T_i} y_i = c(e)$. To make sure the improved y is still feasible, we need to make sure that the constraints are still met. Since $y_k = 0$, we have

that for all $e \in E$, the feasible solution y satisfies

$$\sum_{i \neq k : e \in T_i} y_i \leq c(e)$$

Then the maximum that y_k can be while ensuring that y is still feasible is the distance to the "closest" constraint. In other words,

$$y_k = \min_{e \in T_k} \left\{ c_e - \sum_{i \neq k : e \in T_i} y_i \right\}$$

We then repeat this process until x is a feasible solution. Formally, the primal-dual scheme can be constructed as follows:

Algorithm 4 General Primal-Dual Scheme for MIN-HS

1. Let $y = 0$ and $A = \emptyset$
2. While there exists some $k \in [p]$ such that $A \cap T_k = \emptyset$
3. Increase y_k until there is some $e \in T_k$ such that $\sum_{i : e \in T_i} y_i = c(e)$
4. $A \leftarrow A \cup \{e\}$
5. Output A

To demonstrate how the algorithm works, we give the example below.

Example 2.19. Let $E = \{1, 2, 3, 4, 5\}, T_1 = \{1, 2\}$, $T_2 = \{2, 3\}$, $T_3 = \{1, 4, 5\}$, and lastly define $c(e) = e$ for any $e \in E$. Then, it is easy to see that the optimal hitting set is $A = \{1, 2\}$. By applying the general dual program formulation (2.22) to this instance of MIN-HS, we obtain the following linear program:

$$\begin{array}{ll} \text{maximize} & y_1 + y_2 + y_3 \\ \text{subject to} & y_1 + y_3 \leq 1 \\ & y_1 + y_2 \leq 2 \\ & y_2 \leq 3 \\ & y_3 \leq 4 \\ & y_3 \leq 5 \\ & y_1, y_2, y_3 \geq 0 \end{array} \tag{2.20}$$

Now we begin the algorithm. Start with $y = (0, 0, 0)$ and $A = \emptyset$.

Iteration 1: Since $A \cap T_k = \emptyset$ for all $k \in \{1, 2, 3\}$ currently, suppose we begin by choosing $k = 1$. Note that since y_1 is in the first two constraints of (2.20) and all components of y are zero, the maximum we can increase y_1 to without violating any constraints is $y_1 = 1$. When we do this, the first constraint, which corresponds to $e = 1$, becomes tight. Hence, we add $e = 1$ to A and so $A = \{1\}$.

Iteration 2: Since $1 \in T_1$ and $1 \in T_3$, the only set not hit yet is T_2. Hence, we are only left to choose $k = 2$. Note that since y_2 is in the

second and third constraints of (2.20) and $y_1 = 1$, the maximum we can increase y_2 to without violating any constraints is $y_2 = 1$. When we do this, the second constraint, which corresponds to $e = 2$, becomes tight. Hence, we had $e = 2$ to A and so $A = \{1, 2\}$.

Since all the T_k are hit, we have found an approximate solution to the problem: $A = \{1, 2\}$. In this case, $A(I) = opt(I)$. However, if we chose a different k instead of $k = 1$ in the first iteration, we would have ended up with a non-optimal solution as our approximate solution.

Theorem 2.20. *Let $k = \max_{i\in[p]} |T_i|$. Algorithm 4 is a k-factor approximation for* Min-HS.

Proof.

$$\begin{aligned} A(I) &= \sum_{e\in E} c(e)x_e^A \\ &= \sum_{e\in A} c(e) \end{aligned}$$

But edge e was only added to the set A when the corresponding dual constraint in (2.22) was tight. In other words, $c(e) = \sum_{i:e\in T_e} y_i$, where the y_i are the values of the dual solution after the algorithm ends. Hence,

$$A(I) = \sum_{e\in A} \sum_{i:e\in T_i} y_i$$

We can then rearrange the summation in the above expression

$$\begin{aligned} A(I) &= \sum_{i=1}^{p} \sum_{e:e\in T_i\cap A} y_i \\ &= \sum_{i=1}^{p} |T_i \cap A| y_i \end{aligned}$$

Let $k = \max_{i\in[p]} |T_i|$. Then since $|T_i \cap A| \leq |T_i| \leq k$,

$$\leq k \sum_{i=1}^{p} y_i$$

Recall that $\sum_{i=1}^{p} y_i$ is the objective function of the dual linear program (2.22), and so by the Weak Duality Theorem $\sum_{i=1}^{p} y_i \leq opt$ where opt is the optimal value of the relaxation of (2.18). It follows that

$$\sum_{e\in E} c(e)x_e \leq k \cdot opt$$

and so Algorithm 4 is a k-factor approximation with $k = \max_{i \in [p]} |T_i|$ □

We will now show how we can transform an optimization problem from graph theory into an instance of Min-HS, even if it may not look like one at first. The problem we will use is Minimum Feedback Vertex Set. In order to give the problem definition, we include some preliminaries from graph theory which will be helpful in the construction.

Definition 2.21. Let $G = (V, E)$ be a directed graph. $C \subseteq V$ is called a **feedback vertex set** if the subgraph generated by removing all vertices in C (as well as edges that contain vertices in C) is acyclic.

Definition 2.22. A (directed) graph $G = (V, E)$ is **bipartite** if V can be partitioned into two sets such no two vertices in the same partition are adjacent.

Definition 2.23. Suppose G is a bipartite graph, with $(S, V \setminus S)$ being the partition of vertices. G is a **bipartite tournament** if for any $u \in S$ and $v \in V \setminus S$, either $(u, v) \in E$ or $(v, u) \in E$, but not both.

Example 2.24. Minimum Feedback Vertex Set (Min-FVS): Given a bipartite tournament $G = (V, E)$ with non-negative vertex weight $c : V \to \mathbb{N}$, find a feedback vertex set of minimum weight.

The following Lemma will be the key tool which will connect Min-FVS to Min-HS.

Lemma 2.25. *A bipartite tournament $G = (V, E)$ is acyclic if and only if it contains no cycle of length 4.*

Proof. The forward direction is trivial. If the tournament is acyclic, then in particular, it can not contain any cycles of length 4.

Now suppose that there exists a cycle in G. Let C be a cycle having vertex path $(v_1, v_2, \ldots, v_m, v_1)$ with minimal length m. We will show that $m = 4$ by showing all other possibilities of m give contradictions. Since G is bipartite, we need only consider m that are even. If $m = 2$, then this does not give a valid cycle, as this means that (v_1, v_2) and (v_2, v_1) are both edges in G, which contradicts that G is a bipartite tournament. Suppose that $m \geq 6$. Then since G is a bipartite tournament, there must exist some edge between v_3 and v_m, either (v_3, v_m) or (v_m, v_3), but not both. First suppose that the edge is (v_m, v_3). Then $(v_3, v_4, \ldots, v_m, v_3)$ is a cycle of strictly smaller length, which contradicts that C is minimal. Suppose the edge is (v_3, v_m). Then $(v_1, v_2, v_3, v_m, v_1)$ is a cycle of length 4, which contradicts that C is minimal. Hence, if G is acyclic, there must be no cycles of length 4.

□

Constructing the integer linear program for MIN-FVS will be straightforward if we use Lemma (2.25). Note that in this case, the ground set is V and the hitting sets are all 4-cycles in the graph G. Denote $\mathcal{C}$ the set of all 4-cycles in G. Hence, by adapting the integer linear program (2.22) to MIN-FVS, we have the following integer linear program:

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} c(v) x_v \\ \text{subject to} \quad & \sum_{v \in C} x_v \geq 1 \quad \forall C \in \mathcal{C} \\ & x_v \in \{0, 1\} \end{aligned} \tag{2.21}$$

Now, we find the dual of the relaxation of linear program (2.21).

$$\begin{aligned} \text{maximize} \quad & \sum_{C \in \mathcal{C}} y_C \\ \text{subject to} \quad & \sum_{C : v \in C} y_i \leq c(v) \quad \forall v \in V \\ & y_C \geq 0 \end{aligned} \tag{2.22}$$

We skip writing the primal-dual algorithm for MIN-FVS, as it is essentially the same as Algorithm 4. However, we include the short analysis of the performance of the primal-dual algorithm when adapted to MIN-FVS

Theorem 2.26. *The primal-dual scheme applied to* MIN-FVS *is a* 4*-factor approximation algorithm for* MIN-FVS.

Proof. Note that the hitting sets are 4-cycles, so $\max_{C \in \mathcal{C}} |C| = 4$. By Theorem (2.20), The algorithm is thus a 4-factor approximation. □

References

1. Dinitz, Michael. *601.435/635 Approximation Algorithms*, 2019. https://www.cs.jhu.edu/ mdinitz/classes/ApproxAlgorithms/Spring2019/ Lectures /lecture19.pdf
2. Du, Ding-Zhu, Ker-I Ko, and Xiaodong Hu. *Design and Analysis of Approximation Algorithms*, 62, Springer, New York, 2012.
3. Goemans, Michel X., and David P. Williamson, *The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems, in Approximation Algorithms*, 1997
4. Korte, Bernhard, and Jens Vygen. *Combinatorial Optimization*, Springer, Berlin, 2000.
5. Vazirani, Vijay V. *Approximation Algorithms*, Springer, Berlin, 2011.
6. Zuylen, Anke van. *Linear programming based approximation algorithms for feedback set problems in bipartite tournaments*. Theoretical Computer Science. 412 (2011), 23, 2556-2561.