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 infnite set of combinatorial optimization problems: there is a di erent combinatorial optimization problem for each di erent choice of graph G and cost function c. We would like to de ne a framework that captures this idea.

We say that a class of combinatorial optimization problems is a set of optimization problems that share a common structure and can weaker de nition which is used in practice to analyze the performance of approximation algorithms.

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

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

if is a class of minimization problems, and

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

if is a class of maximization problems.

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

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

This happens to look very similar to the de nition of the performance ratio. Because we are only looking for an upper bound (as opposed to the least upper bound) of the ratios of $A(I)/\mathrm{opt}(I)$, calculating $\mathrm{opt}(I)$

allows one to define a cost function c and a set of linear inequality constraints that will define . 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 to some , where finding the optimum in can be found in polynomial time. The optimal solution in can then be converted, usually through some sort of rounding technique, into a feasible solution in , 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 to a feasible solution in .

To show how a k-factor approximation can be obtained, we will use the following analysis. Suppose an instance I of is the problem $c(x\ ) = \min_{x 2} c(x)$. Now we relax the problem to and let I^0 denote the instance of the problem that gives $c(y\ ) = \min_{y 2} c(y)$. Lastly, we round y to some x^A 2 . We summarize the setup with the figure below:

Then since and we are minimizing over both sets, it follows that $c(y\ )$ $c(x\ )$. Hence

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

To use the notation from the beginning of the introduction, we have that $A(I) = c(x^A)$, $opt(I) = c(x\ )$, and $opt(I^0) = c(y\ )$ and so

$$\frac{A(I)}{opt(I)} \qquad \frac{A(I)}{opt(I^0)} \tag{1.3}$$

Since we are able to obtain $opt(I^0)$ in polynomial time, if we nd a k such that

$$\frac{A(I)}{opt(I^0)} \quad k;$$

for any I, then by (1.3), we have that $A(I)$ k $opt(I)$ and so A is a k-factor approximation for the class of problems . 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 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 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 J/F230E4.x4(p)1(s-,0E4.x4whi Td [(su4.x4wd)-e)-473

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.

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

M3(o Tdr [(M3(h/F43.479670

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 $c : E \to Z^+$ on edges and $k \in Z$, 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

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

j . Wernnec2f 57etermiec2f3.948ecte[(found)-326(a)-327(cut)-8.1d [(n)]TJ/F 11.9552 Tf 169

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

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

or

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

Algorithm 2 Iterated Rounding Approximation for Min-GSN

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

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

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

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

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

$x_e = 0$ 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:

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

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

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

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 (I

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

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

paper. Recall Example (2.1):

$$\begin{array}{ll} \text{minimize} & 3x_1 + x_2 \\ \text{subject to} & 2x_1 + 2x_2 \quad 4 \\ & x_1 \end{array}$$

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

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

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

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

then the dual linear program is

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

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 $c^T x \quad b^T y$.

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

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

Theorem 2.15 (Strong Duality Theorem). The primal problem has a nite optimum i its dual has a nite 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.

De nition 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) $x_j > 0 \Rightarrow (A^T y)_j = c_j$

(2) $y_i > 0 \Rightarrow (Ax)_i = b_i$

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

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

$$\text{maximize} \quad \sum^{p}$$

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 y 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:

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

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 add $e = 2$ 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, $A(I) = \mathrm{opt}(I)$. 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.

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

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

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

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 $\sum_{i=1}^{p} y_i \leq \mathrm{opt}$ where opt is the optimal value of the relaxation of (2.18). It follows that

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

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 C the set of all 4-cycles in G. Hence, by adapting the integer linear program (2.22) to Min-FVS

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 tournamentsTheoretical Computer Science. 412 (2011), 23, 2556-2561.