Integer Programming
ISE 418

Lecture 2

Dr. Ted Ralphs

Reading for This Lecture

- N&W Sections I.1.1-I.1.6
- Wolsey Chapter 1
- CCZ Chapter 2

Formulations and Models

- Our description in the last lecture boiled the modeling process down to two basic steps.

  1. Create a *conceptual model* of the real-world problem.
  2. Translate the conceptual model into a *formulation*.

- In the *conceptual model*, we identify the variables and what values of we would like to allow in logical/conceptual terms.

- In the *formulation*, we specify constraints that ensure the feasible solutions to the resulting mathematical optimization problem are indeed "feasible" in terms of the conceptual model.

- Integer (and other) variables that don't appear in the conceptual model may be introduced to enforce logical conditions (disjunction).

- We also try to account for "solvability."

- We may have to prove formally that the resulting formulation does in fact correspond to the model (and eventually to the real-world problem).

Valid Formulation

- Suppose $\mathcal{F} \subseteq \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$ is a set describing the solutions to our conceptual model.
- Then

$$\mathcal{S} = \left\{(x, y) \in (\mathbb{Z}^p \times \mathbb{R}_+^{n-p}) \times (\mathbb{Z}_+^t \times \mathbb{R}_+^{r-t}) \mid Ax + Gy \leq b\right\}$$

  is a *valid (linear) formulation* if $\mathcal{F} = \mathrm{proj}_x(\mathcal{S})$, where $A \in \mathbb{Q}^{m\times n}, G \in \mathbb{Q}^{r\times n}, b \in \mathbb{Q}^m$ are chosen appropriately.
- The formulation may have auxiliary variables that are not in the conceptual model (we will see an example later in the lecture).
- In fact, the variables from the conceptual model may not even be explicitly needed if their values can be computed later.
- This definition addresses only feasibility and does not address formal equivalence of the formulation and the original optimization problem.
- To prove such equivalence, we need also consider the objective function and may need to invoke the concept of *reduction*, introduced later.

Alternative Formulations

- A typical mathematical model may have many valid formulations.
- In this class, we focus on problems that have linear formulations (naturally, not every problem does).
- We will see that the specific formulation we choose can have a big impact on the efficiency of the solution method.
- Finding a "good" formulation is critical to solving a given linear model efficiently and is a good deal of what this course is about.
- The existence of alternative formulations and the question of how to choose between them will be an implicit theme throughout the course.
- In fact, most algorithms for solving optimization problems can be seen as methods for iteratively reformulating.

Notation and Terminology

- For most parts of the course, we'll assume the formulation is given and won't consider the original conceptual model.
- We may informally refer to the feasible region of the LP relaxation as "the formulation."
- Later we'll discuss mathematical formalities involved in describing optimization problems.
- For ease of notation, we won't distinguish between the original *structural variables* and the additional *auxiliary variables*.

Proving Validity

- There are two parts to proving a formulation is valid, although one or both of these may be "obvious" in some cases.
  - First, we have to prove that $\mathcal{F}$ is in fact the set of solutions to the original problem, which may have been described non-mathematically.
  - Second, we have to prove our formulation is correct.
- In the first step, we need to identify a mapping between the real-world system and the set $\mathcal{F}$.
- Proving validity of a given formulation often means proving $\mathcal{F} = \mathrm{proj}_x(\mathcal{S})$.
- The most straightforward way of doing this involves proving
  - $x \in \mathcal{F} \Rightarrow x \in \mathrm{proj}_x(\mathcal{S})$, and
  - $x \in \mathrm{proj}_x(\mathcal{S}) \Rightarrow x \in \mathcal{F}$.
- Note also that we may need to separately prove that the chosen objective properly ranks the solutions according to our evaluation in the real world.

Problem Reduction

- The process of modeling and formulation involves multiple translations from one formal (or informal) language into another.
- Each of these steps involves what is called *reduction*, a type of procedure that we will study in more detail later in the course.
- Informally, reducing problem A to problem B involves deriving
  - a mapping of each "instance" of problem A to an "instance" of problem B, and
  - a mapping of a solution to problem B to a solution to problem A
- If problem A can be reduced to problem B in this way, we can solve an instance of problem A by
  1. Mapping the instance of problem A to an instance of problem B;
  2. Solving the instance of problem B; and then
  3. Mapping the solution we obtain back to a solution of problem A.
- Note that for an optimization problem, reduction only requires that an *optimal* solution of B maps to an *optimal* solution of A.
- There may be solutions to B that do not map to solutions of A, but also can never be optimal.

Efficient Reduction

- The way reduction was informally described on the previous slide did not account for the difficulty of doing the mapping.
- In general, for a reduction to be useful, the mappings should be "easy" to compute.
- We usually define this to mean that the number of steps required is polynomial in the "size" of the input.
- Hence, the description of the instance of problem B cannot be more than a polynomial factor larger than the input of the instance of problem A.
- We'll define this notion of "efficiency" more formally laster in the course and also study it in ISE 407.
- Also note that we required that problem A be solved by one call to the algorithm for problem B.
- In general, notions of reduction exist in which multiple instances of problem B may be used to solve problem A.
- In this more general notion of reduction, we put a similar limit on both the size and number of instances of problem B to be solved.

Problem Reduction and Modeling

- Note that reduction does not require us to identify a problem that is *equivalent* to our original problem.
- Problems A and B may not be equivalent, since we don't require that every instance of problem B corresponds to an instance of problem A.
- The goal is to exploit an algorithm for problem B to solve problem A.
- Modeling of a general optimization problem involves *reducing* that model to optimization over a set $\mathcal{F}$.
- Proving validity of a formulation amounts to showing that optimization over $\mathcal{F}$ can be reduced to mathematical optimization.
- We may also do reductions from one mathematical optimization problem to another in some cases.
- These reductions may involve problems defined over completely different sets of variables.

Example: Max Independent Set and Max Clique

Formulations with Integer Variables

- From a practical standpoint, what is the purpose of integer variables?

Formulations with Integer Variables

- From a practical standpoint, what is the purpose of integer variables?
- We have seen in the last lecture that integer variable essentially allow us to introduce *disjunctive logic*
- If the variable is associated with a physical entity that is indivisible, then the value must be integer.
  - Product mix problem.
  - Cutting stock problem.
- At its heart, integrality is a kind of disjunctive constraint.
- *0-1 (binary) variables* are often used to formulate more abstract kinds of disjunctions (non-numerical).
  - Formulating yes/no decisions.
  - Enforcing logical conditions.
  - Formulating fixed costs.
  - Formulating piecewise linear functions.

Formulating Binary Choice

- We use binary variables to formulate yes/no decisions.
- Example: Integer knapsack problem
  - We are given a set of items with associated values and weights.
  - We wish to select a subset of maximum value such that the total weight is less than a constant K.
  - We associate a 0-1 variable with each item indicating whether it is selected or not.

$$
\begin{aligned}
\max \ & \sum_{j=1}^{m} c_j x_j \\
\text{s.t.} \ & \sum_{j=1}^{m} w_j x_j \leq K \\
& x \in \{0,1\}^n
\end{aligned}
$$

Formulating Dependent Decisions

- We can also use binary variables to enforce the condition that a certain action can only be taken if some other action is also taken.
- Suppose x and y are binary variables representing whether or not to take certain actions.
- The constraint $x \leq y$ says "only take action x if action y is also taken".

Example: Facility Location Problem

- We are given n potential facility locations and m customers.
- There is a fixed cost c_j of opening facility j.
- There is a cost d_{ij} associated with serving customer i from facility j.
- We have two sets of binary variables.
  - y_j is 1 if facility j is opened, 0 otherwise.
  - x_{ij} is 1 if customer i is served by facility j, 0 otherwise.
- Here is one formulation:

$$
\begin{aligned}
\min \; & \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\
\text{s.t.} \; & \sum_{j=1}^{n} x_{ij} = 1 & \forall i \\
& \sum_{i=1}^{m} x_{ij} \leq m y_j & \forall j \\
& x_{ij}, y_j \in \{0, 1\} & \forall i, j
\end{aligned}
$$

Selecting from a Set

- We can use constraints of the form $\sum_{j \in T} x_j \geq 1$ to represent that at least one item should be chosen from a set T.
- Similarly, we can also formulate that at most one or exactly one item should be chosen.
- Example: Set covering problem
  - A set covering problem is any problem of the form

$$\begin{aligned} &\min c^\top x \\ &\text{s.t. } Ax \geq 1 \\ &\qquad x_j \in \{0,1\}\ \forall j \end{aligned}$$

    where A is a 0-1 matrix.
  - Each row of A represents an item from a set S.
  - Each column A_j represents a subset S_j of the items.
  - Each variable x_j represents selecting subset S_j.
  - The constraints say that $\cup_{\{j | x_j = 1\}} S_j = S$.
  - In other words, each item must appear in at least one selected subset.

Formulating Disjunctive Constraints

- We are given two constraints $a^\top x \geq b$ and $c^\top x \geq d$ with non-negative coefficients.
- Instead of insisting both constraints be satisfied, we want at least one of the two constraints to be satisfied.
- To formulate this, we define a binary variable y and impose

$$\begin{aligned} a^\top x &\geq yb, \\ c^\top x &\geq (1-y)d, \\ y &\in \{0,1\}. \end{aligned}$$

- More generally, we can impose that at least k out of m constraints be satisfied with

$$(a_i')^\top x \geq b_i y_i, \quad i \in [1..m]$$
$$\sum_{i=1}^{m} y_i \geq k,$$
$$y_i \in \{0,1\}$$

Formulating a Restricted Set of Values

- We may want variable x to only take on values in the set $\{a_1, \ldots, a_m\}$.
- We introduce m binary variables $y_j, j = 1, \ldots, m$ and the constraints

$$x = \sum_{j=1}^{m} a_j y_j,$$

$$\sum_{j=1}^{m} y_j = 1,$$

$$y_j \in \{0, 1\}$$

Piecewise Linear Cost Functions

- We can use binary variables to formulate arbitrary piecewise linear cost functions.
- The function is specified by ordered pairs $(a_i, f(a_i))$ and we wish to evaluate it at a point x.
- We have a binary variable y_i, which indicates whether $a_i \leq x \leq a_{i+1}$.
- To evaluate the function, we take linear combinations $\sum_{i=1}^{k} \lambda_i f(a_i)$ of the given functions values.
- This only works if the only two nonzero $\lambda_i' s$ are the ones corresponding to the endpoints of the interval in which x lies.

Minimizing Piecewise Linear Cost Functions

- The following formulation minimizes the function.

$$
\begin{aligned}
\min \sum_{i=1}^{k} & \lambda_i f(a_i) \\
\text{s.t. } \sum_{i=1}^{k} & \lambda_i = 1, \\
& \lambda_1 \le y_1, \\
& \lambda_i \le y_{i-1} + y_i, \quad i \in [2..k-1], \\
& \lambda_k \le y_{k-1}, \\
& \sum_{i=1}^{k-1} y_i = 1, \\
& \lambda_i \ge 0, \\
& y_i \in \{0, 1\}.
\end{aligned}
$$

- The key is that if $y_j = 1$, then $\lambda_i = 0, \ \forall i \ne j, j+1$.

Formulating General Nonconvex Functions

- One way of dealing with general nonconvexity is by dividing the domain of a nonconvex function into regions over which it is convex (or concave).
- We can do this using integer variables to choose the region.
- This is precisely what is done in the case of the piecewise linear cost function above.
- Most methods of general global optimization use some form of this approach.

Fixed-charge Problems

- In many instances, there is a fixed cost and a variable cost associated with a particular decision.
- Example: Fixed-charge Network Flow Problem
  - We are given a directed graph $G = (N, A)$.
  - There is a fixed cost c_{ij} associated with "opening" arc (i, j) (think of this as the cost to "build" the link).
  - There is also a variable cost d_{ij} associated with each unit of flow along arc (i, j).
  - Consider an instance with a single supply node.
    - Minimizing the fixed cost by itself is a minimum spanning tree problem (easy).
    - Minimizing the variable cost by itself is a minimum cost network flow problem (easy).
    - We want to minimize the sum of these two costs (difficult).

Formulating the Fixed-charge Network Flow Problem

- To formulate the FCNFP, we associate two variables with each arc.
  - x_{ij} (*fixed-charge variable*) indicates whether arc (i,j) is open.
  - f_{ij} (*flow variable*) represents the flow on arc (i,j).
  - Note that we have to ensure that $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

$$
\begin{array}{lll}
\min & \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} + d_{ij}f_{ij} & \\
\text{s.t.} & \displaystyle\sum_{j\in O(i)} f_{ij} - \sum_{j\in I(i)} f_{ji} = b_i & \forall i \in N \\
& f_{ij} \le Cx_{ij} & \forall (i,j) \in A \\
& f_{ij} \ge 0 & \forall (i,j) \in A \\
& x_{ij} \in \{0,1\} & \forall (i,j) \in A
\end{array}
$$