Advanced Operations Research Techniques
IE316

Lecture 21

Dr. Ted Ralphs

Reading for This Lecture

- Bertsimas Sections 10.1, 11.4

Integer Linear Programming

- An *integer linear program* (ILP) is the same as a linear program except that the variables can take on only integer values.
- If only some of the variables are constrained to take on integer values, then we call the program a *mixed integer linear program* (MILP).
- The general form of a MILP is

$$\begin{array}{rl} min & c^T x + d^T y \\ s.t. & Ax + By = b \\ & x, y \geq 0 \\ & x \text{ integer} \end{array}$$

- We have already seen a number of examples of integer programs.
  - Product mix problem
  - Cutting stock problem
  - Integer knapsack problem
  - Assignment problem
  - Minimum spanning tree problem

How Hard is Integer Programming?

- Solving general integer programs can be much more difficult than solving linear programs.
- There in no known *polynomial-time* algorithm for solving general MILPs.
- Solving the associated *linear programming relaxation* results in a lower bound on the optimal solution to the MILP.
- In general, an optimal solution to the LP relaxation does not tell us anything about an optimal solution to the MILP.
  - Rounding to a feasible integer solution may be difficult.
  - The optimal solution to the LP relaxation can be arbitrarily far away from the optimal solution to the MILP.
  - Rounding may result in a solution far from optimal.
  - We can bound the difference between the optimal solution to the LP and the optimal solution to the MILP (how?).

Duality in Integer Programming

- Let's consider again an integer linear program

$$\begin{aligned} min \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \\ & x \text{ integer} \end{aligned}$$

- As in linear programming, there is a duality theory for integer programs.
- We can "dualize" some of the constraints by allowing them to be violated and then penalizing their violation in the objective function.
- We relax some of the constraints by defining, for given Lagrange multipliers p, the Lagrangean relaxation

$$Z(p) = \min_{x \in X} \{c^T x + p^T (A' x - b)\}$$

where $X = \{x \in \mathbb{Z}^n | A'' x = b, x \geq 0\}$ and $A^T = [(A')^T, (A'')^T]$.

More Integer Programming Duality

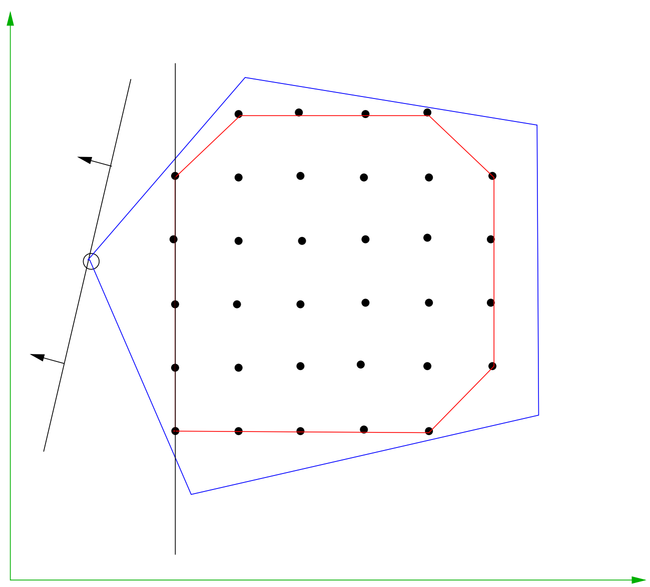
- $Z(p)$ is a lower bound on the optimal solution to the original ILP, so we consider the *Lagrangean dual* $\max Z(p)$.
- As long as we can optimize over the set X, we can solve the Lagrangean dual efficiently.
- As before, the optimal solution to the Lagrangean dual yields a lower bound on the optimal value of the original ILP (weak duality).
- However, for integer programming, strong duality does not hold.
- The difference between the optimal solution to the ILP and the optimal solution to the dual is called the *duality gap*.
- This is another indication of why integer programming is difficult.

The Geometry of Integer Programming

- Let's consider again an integer linear program

$$\begin{aligned} min \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \\ & x \text{ integer} \end{aligned}$$

- The feasible region is the integer points inside a polyhedron.



- It is easy to see why solving the LP relaxation does not necessarily yield a good solution.

Easy Integer Programs

- As we have already seen, certain integer programs are "easy".
- What makes an integer program "easy"?
  - All of the extreme points of the LP relaxation are integral.
  - Every square submatrix of A has determinant $+1$, -1, or 0.
  - We know a *complete description* of the convex hull of feasible solutions.
  - We have an efficient algorithm for finding an optimal integer solution (other than linear programming).
  - There is no duality gap.
- Examples of "easy" integer programs.
  - Minimum cost network flow problems.
  - Assignment problem.
  - Minimum cost spanning tree problem.

Modeling with Integer Variables

- Why do we need integer variables?
- We have already seen some examples.
- If the variable is associated with a physical entity that is indivisible, then it must be integer.
  - Product mix problem.
  - Cutting stock problem.
- We can use *0-1 (binary) variables* for a variety of purposes.
  - Modeling yes/no decisions.
  - Enforcing disjunctions.
  - Enforcing logical conditions.
  - Modeling fixed costs.
  - Modeling piecewise linear functions.

Modeling Binary Choice

- We use binary variables to model 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.

$$
\begin{aligned}
max \ & \sum_{j=1}^{m} c_j x_j \\
s.t. \ & \sum_{j=1}^{m} w_j x_j \leq K \\
& x \geq 0 \\
& x \quad integer
\end{aligned}
$$

Modeling 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 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 that must be serviced from those locations.
- 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.

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

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 model 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

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

    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.

Example: Combinatorial Auctions

- The winner determination problem for a *combinatorial auction* is a set covering problem.
- The rows represent items or services that a buyer is trying to acquire.
- The columns represent subsets of the items that a particular supplier can provide for a specified cost.
- The object is to select a subset of the bidders such that
  - cost is minimized, and
  - every item is provided by at least one bidder.
- This is a set covering problem.
- Similarly, we can also consider set packing and set partitioning problems.

Modeling Disjunctive Constraints

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

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

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

$$(a_i')^T x \geq b_i y_i, \quad i \in [1..m]$$

$$\sum_{i=1}^{m} y_i \geq k,$$

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

Modeling 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

$$
\begin{aligned}
x &= \sum_{j=1}^{m} a_j y_j, \\
\sum_{j=1}^{m} y_j &= 1, \\
y_j &\in \{0, 1\}
\end{aligned}
$$

Piecewise Linear Cost Functions

- We can use binary variables to model 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 will 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.

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

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

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).
  - 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).

Modeling the Fixed-charge Network Flow Problem

- To model 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 $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

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