CE 205A
Transportation Logistics

Lecture 12

Branch and Cut

Previously on Transportation Logistics

Suppose you run logistics company. You can open your offices at any of n potential locations in the city and there is a fixed cost of opening a branch at node j, which is denoted by f_j.

You can serve customer demand at m locations from any of the branches. The cost of serving customer at i from branch j is c_{ij}. Where should you open branches and how do you pair customers to branches?

$$\min \sum_{j=1}^{n} f_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i = 1, \ldots, m$$

$$x_{ij} \leq y_j \qquad \forall\, i = 1, \ldots, m, j = 1, \ldots, n$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i = 1, \ldots, m, j = 1, \ldots, n$$

$$y_j \in \{0, 1\} \qquad \forall\, j = 1, \ldots, n$$

Previously on Transportation Logistics

In the facility location problem (say P_1), we set $x_{ij} \leq y_j$ to indicate that a customer at i can be paired to a branch at j only if it is open, i.e., $y_j = 1$.

Alternately, we can add all such constraints for $i = 1, \ldots, m$ and write a model P_2 for which

$$\sum_{i=1}^{m} x_{ij} \leq m y_j \qquad \forall j = 1, \ldots, n$$

Which of the two formulations is better? P_2 has fewer constraints than P_1, but $P_1 \subset P_2$. Imagine $x_1 \leq 1$ and $x_2 \leq 1$ vs. $x_1 + x_2 \leq 2$. (Although the integer solutions in these two examples is not the same.)

Hence, if you solve the LP relaxations of the two problems, you may notice that $z^{LP}_{P_1} \geq z^{LP}_{P_2}$.

Previously on Transportation Logistics

Show that $x_i \leq y$ is a valid inequality and a facet of the polytope

$$X = \left\{ (\mathbf{x}, y) \in \mathbb{R}^m_+ \times \{0,1\} : \sum_{i=1}^{m} x_i \leq my, x_i \leq 1, i = 1, \ldots, m \right\}$$

The required result can be shown by proving the following statements.

- $\dim(\text{Conv}(X)) = m + 1$
- $F_i = \left\{ (\mathbf{x}, y) \in Conv(X) : x_i = y \right\}$ is a facet, i.e., $\dim(F_i) = m$.

Previously on Transportation Logistics

The ideas used in column generation can also be applied to problems with a few variables but a large number of constraints.

Consider the dual of the standard form as shown below.

$$\begin{aligned} \max \ & \mathbf{b}^{\mathsf{T}}\mathbf{y} \\ \text{s.t.} \ & \mathbf{A}^{\mathsf{T}}\mathbf{y} \leq \mathbf{c} \end{aligned}$$

Instead of solving the complete problem, we solve a *relaxed* version of it with fewer constraints.

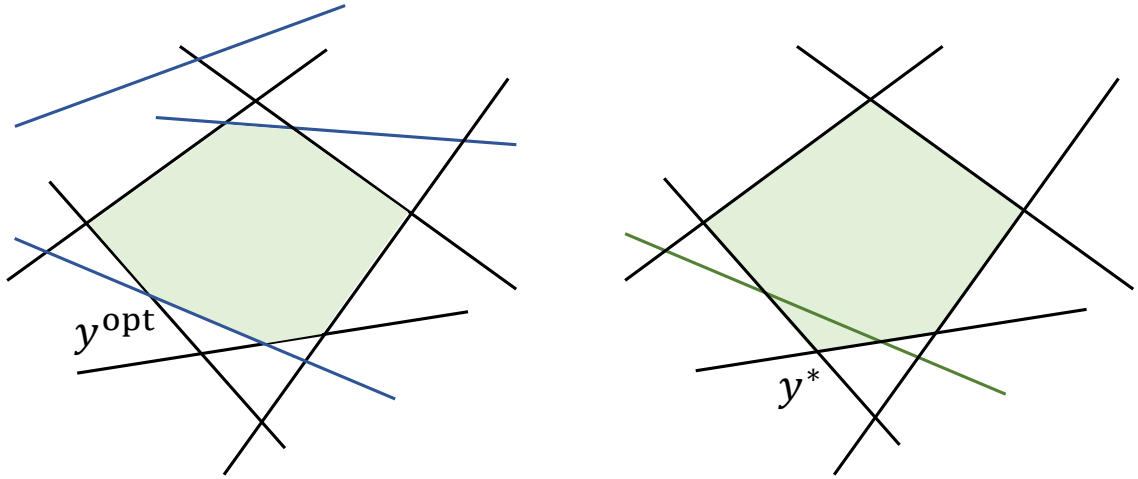
$$\begin{aligned} \max \ & \mathbf{b}^{\mathsf{T}}\mathbf{y} \\ \text{s.t.} \ & \mathbf{A}_{\cdot j}^{\mathsf{T}}\mathbf{y} \leq c_j \, \forall \, j \in J \end{aligned}$$

If the optimal solution to this problem solves all the left out constraints then then it is optimal to the original problem.

However, if some constraint is violated, we add it to the relaxed problem and resolve. How can we identify constraints that are violated easily?

Previously on Transportation Logistics

To identify the violating constraint, we solve a *separation problem* in which $\mathbf{y}^*$ is separated from the original feasible region using a constraint which it violates just as done in cutting plane methods.



This constraint can be identified by solving $\min c_j - \mathbf{A}_{\cdot j}^{\mathsf{T}} \mathbf{y}^* \; \forall \, j \in J^c$. What are the decision variables in this problem? If the optimal solution to this problem is ≥ 0, $\mathbf{y}^*$ is optimal to the original problem.

Previously on Transportation Logistics

Consider an IP formulation $X = \{\mathbf{x} \in \mathbb{Z}^n_+ : \mathbf{Ax} \leq \mathbf{b}\}$. Let Conv$(X)$ be the convex hull of the feasible space and $P = \{\mathbf{x} \in \mathbb{R}^n_+ : \mathbf{Ax} \leq \mathbf{b}\}$ be the LP relaxation space.

We can try to cut off the current solution to the LP relaxation using a specific valid inequality so that we move closer to Conv(X).

Algorithm 1 Generic Cutting Plane Algorithm

$\mathbf{x}^* \leftarrow \arg\min_{\mathbf{x} \in P} \mathbf{c}^\mathsf{T}\mathbf{x}$

while $\mathbf{x}^* \notin \mathbb{Z}^n_+$ **do**

Separation Problem:

Determine a valid inequality $(\mathbf{w}, w_0)$ that satisfies $\mathbf{w}^\mathsf{T}\mathbf{x} > w_0$

$P \leftarrow P \cap \{\mathbf{w}^\mathsf{T}\mathbf{x}^* > w_0\}$

$\mathbf{x}^* \leftarrow \arg\min_{\mathbf{x} \in P} \mathbf{c}^\mathsf{T}\mathbf{x}$

end while

Does $\mathbf{c}^\mathsf{T}\mathbf{x}$ always increase?

Lecture Outline

1. Branch and Cut
2. CPLEX Example

Lecture Outline

Branch and Cut

Branch and Cut

Introduction

The cutting plane method described in the previous lectures may perform well if they are facets. Else, they tend to exhibit marginal improvement in successive iterations.

Furthermore, ever cut increases the time required to solve the LPs. Hence, it is prudent to branch if there is no reasonable improvement in the objective function.

One can generate new cuts at nodes other than the root in the branch and bound tree, which gives rise to the branch and cut method.

Several decisions affect the computational performance of such methods.

- When should you branch in the branch and bound tree?
- What kind of cuts should be added?
- How much time should we spend in solving separation problems?
- Can we implement separation heuristics to generate cuts?
- Do we retain the cuts added at ancestor nodes? If not, which ones should be dropped and when? (slacks are positive or duals are non-zero)

Branch and Cut

Introduction

Most libraries tend to have a *cut generator* module which automatically determines "good" cuts from typical families of cuts such as covers, cliques, MIR, Gomory, etc.

They then use these in *local and global cut pools*. Some or all of the cuts in these pools may be added depending on the solver. The ones that are promising in the local pool are used longer (at other nodes) by moving them to the global cut pool.

The exact implementation details are usually proprietary except for open source solvers. See the following reference for more details.

Ladányi, L., Ralphs, T. K., & Trotter, L. E. (2001). Branch, cut, and price: Sequential and parallel. In Computational combinatorial optimization (pp. 223-260). Springer, Berlin, Heidelberg.

Branch and Cut

Example

Solve the following optimization problem using branch and bound and branch and cut while adding at most one cover inequality at each node.

$$
\begin{aligned}
\min \quad & -8x_1 - 22x_2 - 10x_3 - 15x_4 \\
\text{s.t.} \quad & x_1 + 4x_2 + 7x_3 + 5x_4 \leq 11 \\
& 0 \leq x_1 \leq 4 \\
& 0 \leq x_2 \leq 1 \\
& 0 \leq x_3 \leq 1 \\
& 0 \leq x_4 \leq 1
\end{aligned}
$$

How many nodes are explored in both cases?

CPLEX Example

CPLEX Example

Outline

Consider the facility location model as shown below where J and C are the sets of facilities and clients, respectively.

The constraints are aggregated with a minor difference that a facility can serve at most $|C| - 1$ clients. That is, more than one facility must be opened.

$$\min \sum_{j \in J} \texttt{fixedCost[j]} * \texttt{used[j]} + \sum_{j \in J} \sum_{c \in C} \texttt{cost[c][j]} * \texttt{supply[c][j]}$$

$$\text{s.t.} \sum_{j \in J} \texttt{supply[c][j]} = 1 \qquad \forall c \in C$$

$$\sum_{c \in C} \texttt{supply[c][j]} \leq (|C| - 1) * \texttt{used[j]} \qquad \forall j \in J$$

$$\texttt{supply[c][j]} \in \{0, 1\} \qquad \forall c \in C, j \in J$$

$$\texttt{used[j]} \in \{0, 1\} \qquad \forall j \in J$$

CPLEX Example

Outline

It is easy to integrate customized cuts for MIP problems with standard solvers.

Most solvers provide features called callbacks which direct them to alternate user-specified strategies during the branch and bound procedure.

The following discussion is specific to CPLEX 22.1.0.0.

Functions:

- `main()`: Reads arguments and data and calls `admipex8()`
- `usage()`: Specifies the arguments of the code (multiple arguments are allowed)
- `admipex8()`: Contains the main formulation and initialization of the cuts

CPLEX Example

Outline

Callbacks are specified using classes. `FacilityCallback` is the class that generates the cuts and `facilitycb` is an instance/object of the class.

- `clients`: List of client indices
- `locations`: List of location indices
- `used`: CPLEX variable iterator
- `supply`: CPLEX variable iterator
- `cutlhs`: List of $|J| \times |C|$ disaggregate constraint expressions
- `cutrhs`: List of $|J| \times |C|$ zeros

Unlike simple CPLEX programs, the variable iterators must be defined and used in the constructor for the callback class.

CPLEX Example

Outline

Class functions:

- `disaggregate()`
- `cuts_from_table()`
- `lazy_capacity()`
- `invoke()`

Each of the first three implements a specific type of cuts. The fourth function tells CPLEX which one of these three to use.

These functions use the current solution which is stored in the class objects. However, since the problem is not fully solved one cannot access `cpx.solution.get_values()`

Instead, depending on the type of cuts added, `context.get_candidate_point()` and `context.get_relaxation_point()` can be used.

CPLEX Example

Lazy Cuts

Lazy cuts refer to constraints in the original formulation that may be skipped so that the optimization model can run faster LPs with fewer constraints. These constraints are not specified in the original model.

For example, if `-lazy` is passed as an argument, the constraint

$$\sum_{c \in C} \texttt{supply[c][j]} \leq (|C| - 1) * \texttt{used[j]}$$

is not added upfront, but is only included if it is violated.

If the intermediate LP solution is integral when lazy cuts are included, CPLEX will not terminate until the solution satisfies all the left out constraints.

CPLEX Example

User Cuts

Lazy cuts are similar to row generation methods and work well when there are an exponential number of cuts to choose from.

For regular cutting plane methods, we can provide cuts that are known to be valid inequalities either by directly providing an entire family of inequalities or by solving a separation problem.

These are also called *user cuts*. In the example, `disaggregate()` adds the facet-defining inequality of the type `supply[c][j]` $\leq$ `used[j]`

Note that there are subtle differences in declaring the user and lazy cuts `context.add_user_cut()` and `context.reject_candidate()`. The keyword arguments are also slightly different.

CPLEX Example

User Cuts

`cuts_from_table()` is a variant of the disaggregate user cuts in which the cut expressions are stored upfront in a table and are added if the LHS is greater than the RHS.

CPLEX allows you to configure a few additional options for user cuts.

- You can add cuts only along a subtree rooted at the current node. This can be set by turning the `local` variable to True.
- Cuts can also be removed at later stages and CPLEX can be asked to decide it internally using the `cutmanagement` option. Use `purge` to dynamically delete cuts and `force` to retain them once added.

CPLEX Example

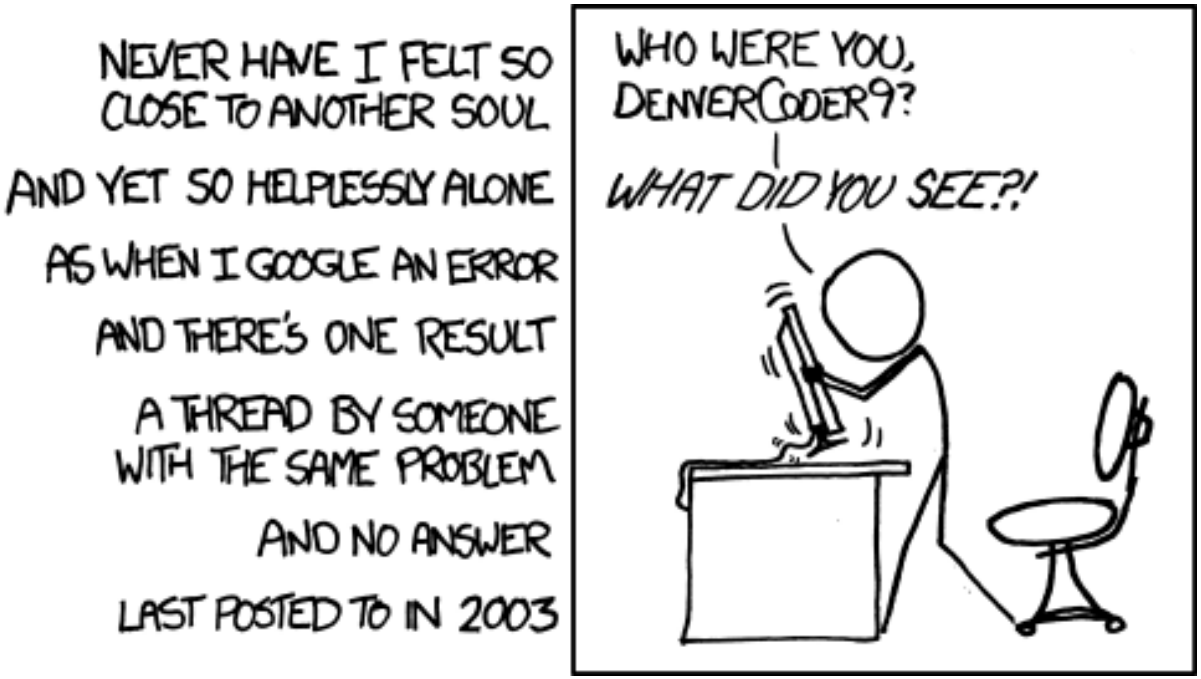
Invoking Cuts

The `contextmask` variable in `admipex8()` tells the `cpx.set_callback()` function the exact situations in which callbacks must be invoked.

These are declared using the bitwise OR operators | (e.g., 00111 | 10010 = 10111) on the context ID.

The class variable `candidate` is used for lazy cuts and `relaxation` is used for user cuts.

Your Moment of Zen



Source: xkcd.com