Advanced Operations Research Techniques
IE316

Lecture 16

Dr. Ted Ralphs
Reading for This Lecture

• Bertsimas 6.1-6.3
Large-scale Linear Programming

• Linear programs occurring in practice can be extremely large.

• For large LPs, the vast majority of the matrix is superfluous.

• To solve a particular instance, we only need
  – Constraints that are binding at optimality.
  – Variables that are basic at optimality.

• In fact, we really only need constraints that have positive dual values
  and variables that have positive values at optimality.

• If we only had these variables and constraints from the start, we could
  solve very easily.

• The problem is that we don’t know which variables and constraints these
  are.
Delayed Column Generation

- In problems with large numbers of variables, there are two main difficulties.
  - The time required just to generate the matrix.
  - The time required to calculate the reduced costs each iteration.

- We can address both of these problem with delayed column generation.

- Idea
  - Start with a subset of “promising” columns.
  - Solve the LP with just these columns.
  - Price the remaining columns and add those with negative reduced costs.
  - Iterate.
Automatic Delayed Column Generation

- In fact, we only need to find the column with the most negative reduced cost.

- This is an optimization problem!

- If we can solve this optimization problem, then we can solve the LP without explicitly listing the columns.

- There are many variants of this basic algorithm.

- All are based in the ability to generate a column with negative reduced cost, given the current dual prices.
**Generic Column Generation Algorithm**

- We are interested in solving an LP with a large number of columns.
- Consider the *restricted problem* obtained by considering only the subset of the columns indexed by set $I$.

$$
\begin{align*}
\min & \sum_{i \in I} c_i x_i \\
\text{s.t.} & \sum_{i \in I} A_i x_i = b \\
& x \geq 0
\end{align*}
$$

- Solve this LP and calculate the optimal dual solution.
- Now, we must generate a new column $A_j$ for which $c_j - c_B B^{-1} A_j < 0$.
- This can be done by solving the *column generation subproblem*

$$
\min_{a \in C} c_a - c_B B^{-1} a,
$$

where $C$ is the global set of columns.
- If the minimum is $< 0$, add the new column to the set $I$ and re-solve.
Maintaining the Restricted Problem

- Many variants of this algorithm can be obtained by changing how the set $I$ is maintained.

- One obvious variant is to simply maintain every column that has been generated so far in $I$.

- Another variant is to throw away all the nonbasic columns after each iteration.

- There are many intermediate options.
Example: The Cutting Stock Problem

- A prototypical problem that can be solved by column generation is the cutting stock problem from the homework.
- The large rolls from which the smaller rolls are cut have width $W$.
- The desired widths of the smaller rolls is represented by vector $w \in \mathbb{R}^m$.
- In this problem, potential columns correspond to feasible patterns.
- A given column vector $a$ corresponds to a feasible pattern if and only if

$$\sum_{i=1}^{m} a_i w_i \leq W$$

and $a$ contains only nonnegative integers.
The Column Generation Subproblem

• The cost of every pattern (column) is identical.

• Hence, the column generation subproblem is

\[
\begin{align*}
\text{max} & \quad \sum_{i=1}^{m} p_i a_i \\
\text{s.t.} & \quad \sum_{i=1}^{m} w_i a_i \leq W \\
& \quad a_i \geq 0 \\
& \quad a_i \text{ integer}
\end{align*}
\]

• This problem is known as the \textit{integer knapsack problem}.

• Think of a shopping spree in which you try to maximize the value of the set of items that will fit into a shopping cart.

• This problem can be solved using \textit{dynamic programming}.
Finding an Initial BFS

• For this problem, an initial BFS is easy to find.

• Take the $i^{th}$ basic column to be the $i^{th}$ unit vector, i.e., the pattern obtained by cutting one roll of width $w_i$.

• This set of columns forms an initial feasible basis.

• Of course, these columns are not likely to be used in an optimal solution.
Algorithm Summary

• Construct the initial BFS and add these columns to the set $I$.
• Solve the restricted LP and calculate the optimal dual solution.
• Solve the column generation subproblem (CGS).
• If the optimal solution to the CGS has negative cost, then add the new column to $I$ and iterate.
• Otherwise, the current solution is optimal.
Constraint Generation Methods

• We can use the same methodology to solve problems with large numbers of constraints.

• Again, recall that we only really “need” the constraints that are binding at optimality.

• Actually, we only need the constraints whose corresponding dual variable has nonzero value at optimality.

• Keeping unneeded constraints in the formulation causes the size of the basis to increase.

• The size of the basis is one of the biggest determinants of the speed of the simplex algorithm.
Development of the Method

- We wish to solve an LP with a large number of rows.
- The method is the same as for column generation, except that now we solve the problem on a restricted row set.
- We attempt to generate an inequality from the global set that is violated by the current optimal solution.
- This is called the separation problem because it is the problem of separating the current solution from the polyhedron with a hyperplane.
- Note that in the dual, this method is a column generation method, so we have already developed all the machinery we need.
Generic Constraint Generation Algorithm

- Consider the *restricted problem* obtained by considering only the subset of the rows indexed by set $I$.

\[
\begin{align*}
\min & \quad c^T x \\
\text{s.t.} & \quad a_i^T x \geq b \quad \forall i \in I
\end{align*}
\]

- Solve this LP and calculate the optimal primal solution $\hat{x}$.

- Now, we must generate a new row $a_j$ for which $b_j - a_j \hat{x} < 0$.

- This can be done by solving the *constraint generation subproblem*

\[
\min_{\substack{a \in R}} \left( c_a a^T x \right),
\]

where $R$ is the global set of constraints.

- Add the new constraint to the set $I$ and re-solve.
Maintaining the Restricted Problem

• Again, many variants of this algorithm can be obtained by changing how the set $I$ is maintained.

• One obvious variant is to simply maintain every constraint that has been generated so far in $I$.

• Another variant is to throw away all the nonbinding constraints after each iteration.

• There are many intermediate options.