Linear Optimization: Column Generation

Brady Hunsaker

November 22, 2004

Column Generation and the Pricing Problem

- If a problem has many variables/columns but relatively few constraints, then column generation may be beneficial.

Column Generation and the Pricing Problem

- If a problem has many variables/columns but relatively few constraints, then column generation may be beneficial.
- In order to be practical, we must be able to solve the *pricing problem*: given a set of dual values, either identify a column that has a favorable reduced cost, or indicate that no such column exists.

Column Generation and the Pricing Problem

- If a problem has many variables/columns but relatively few constraints, then column generation may be beneficial.
- In order to be practical, we must be able to solve the *pricing problem*: given a set of dual values, either identify a column that has a favorable reduced cost, or indicate that no such column exists.
- Recall from the simplex algorithm in matrix notation:

Column Generation and the Pricing Problem

- If a problem has many variables/columns but relatively few constraints, then column generation may be beneficial.
- In order to be practical, we must be able to solve the *pricing problem*: given a set of dual values, either identify a column that has a favorable reduced cost, or indicate that no such column exists.
- Recall from the simplex algorithm in matrix notation:
  - The dual values associated with a solution are given by the vector $y^T = c_B^T B^{-1}$

Column Generation and the Pricing Problem

- If a problem has many variables/columns but relatively few constraints, then column generation may be beneficial.
- In order to be practical, we must be able to solve the *pricing problem*: given a set of dual values, either identify a column that has a favorable reduced cost, or indicate that no such column exists.
- Recall from the simplex algorithm in matrix notation:
  - The dual values associated with a solution are given by the vector $y^T = c_B^T B^{-1}$
  - The reduced cost of a column A_j is given by $c_j - c_B^T B^{-1} A_j = c_j - y^T A_j$

Column Generation and the Pricing Problem

- If a problem has many variables/columns but relatively few constraints, then column generation may be beneficial.
- In order to be practical, we must be able to solve the *pricing problem*: given a set of dual values, either identify a column that has a favorable reduced cost, or indicate that no such column exists.
- Recall from the simplex algorithm in matrix notation:
  - The dual values associated with a solution are given by the vector $y^T = c_B^T B^{-1}$
  - The reduced cost of a column A_j is given by $c_j - c_B^T B^{-1} A_j = c_j - y^T A_j$
- So given y, we must determine whether there exists a column A_j such that $y^T A_j$ is favorable.

Cutting-stock example

- We will demonstrate column generation with a classical example called a *cutting stock problem*.

Cutting-stock example

- We will demonstrate column generation with a classical example called a *cutting stock problem*.
- When paper is produced, it is generally produced in large rolls with a fixed width. These are called *raw rolls* or *raws*.

Cutting-stock example

- We will demonstrate column generation with a classical example called a *cutting stock problem*.
- When paper is produced, it is generally produced in large rolls with a fixed width. These are called *raw rolls* or *raws*.
- Customers desire to have rolls in a variety of smaller widths, so each raw is cut (by special tools) into several smaller *final rolls*.

Cutting-stock example

- We will demonstrate column generation with a classical example called a *cutting stock problem*.
- When paper is produced, it is generally produced in large rolls with a fixed width. These are called *raw rolls* or *raws*.
- Customers desire to have rolls in a variety of smaller widths, so each raw is cut (by special tools) into several smaller *final rolls*.
- Typically there is some "leftover" part of the raw that is wasted.

Cutting-stock example

- We will demonstrate column generation with a classical example called a *cutting stock problem*.
- When paper is produced, it is generally produced in large rolls with a fixed width. These are called *raw rolls* or *raws*.
- Customers desire to have rolls in a variety of smaller widths, so each raw is cut (by special tools) into several smaller *final rolls*.
- Typically there is some "leftover" part of the raw that is wasted.
- The cutting-stock problem is as follows: given a raw size and a set of demands for final sizes, determine how to cut the raws so that the fewest number of raw rolls are used.

Cutting-stock example

- We will demonstrate column generation with a classical example called a *cutting stock problem*.
- When paper is produced, it is generally produced in large rolls with a fixed width. These are called *raw rolls* or *raws*.
- Customers desire to have rolls in a variety of smaller widths, so each raw is cut (by special tools) into several smaller *final rolls*.
- Typically there is some "leftover" part of the raw that is wasted.
- The cutting-stock problem is as follows: given a raw size and a set of demands for final sizes, determine how to cut the raws so that the fewest number of raw rolls are used.
- This is equivalent to minimizing the total waste.

Formulating a cutting-stock instance

Formulating a cutting-stock instance

- Create a variable for each possible *pattern* that may be cut from a raw roll. The value of the variable tells how many raw rolls to cut in that pattern.

Formulating a cutting-stock instance

- Create a variable for each possible *pattern* that may be cut from a raw roll. The value of the variable tells how many raw rolls to cut in that pattern.
- Let P be the set of patterns, I be the set of final roll sizes, d_i be the demand for final $i \in I$, and a_{ip} be the number of size i finals in pattern p.

Formulating a cutting-stock instance

- Create a variable for each possible *pattern* that may be cut from a raw roll. The value of the variable tells how many raw rolls to cut in that pattern.
- Let P be the set of patterns, I be the set of final roll sizes, d_i be the demand for final $i \in I$, and a_{ip} be the number of size i finals in pattern p.
- $$\begin{array}{lrcll} \min & \sum_{p \in P} x_p & & & \\ s.t. & \sum_{p \in P} a_{ip} x_p & \geq & d_i & \forall i \in I \\ & x_p & \geq & 0 & \end{array}$$

Formulating a cutting-stock instance

- Create a variable for each possible *pattern* that may be cut from a raw roll. The value of the variable tells how many raw rolls to cut in that pattern.
- Let P be the set of patterns, I be the set of final roll sizes, d_i be the demand for final $i \in I$, and a_{ip} be the number of size i finals in pattern p.
- $$\begin{array}{llcll} \min & \sum_{p \in P} x_p & & & \\ s.t. & \sum_{p \in P} a_{ip} x_p & \geq & d_i & \forall i \in I \\ & x_p & \geq & 0 & \end{array}$$
- Really we should have integer values, but in practice the numbers are mostly large enough that rounding gives a good solution.

Formulating the pricing problem

We need to find values a_i for $i \in I$ such that the values form a feasible pattern and the reduced cost is favorable.

Solving the pricing problem

- An efficient way to solve this subproblem is to use *dynamic programming*.

Solving the pricing problem

- An efficient way to solve this subproblem is to use *dynamic programming*.
- We mentioned dynamic programming before, since it is also an efficient way to solve many shortest-path instances.

Dantzig-Wolfe Decomposition

- A common use of column generation is to separate a set of "easy" constraints from the rest of a large LP. This is called Dantzig-Wolfe decomposition.

Dantzig-Wolfe Decomposition

- A common use of column generation is to separate a set of "easy" constraints from the rest of a large LP. This is called Dantzig-Wolfe decomposition.
- There are many reasons why the "easy" constraints may be easier than the full formulation:

Dantzig-Wolfe Decomposition

- A common use of column generation is to separate a set of "easy" constraints from the rest of a large LP. This is called Dantzig-Wolfe decomposition.
- There are many reasons why the "easy" constraints may be easier than the full formulation:
  - The easy constraints may involve only a small fraction of the variables.

Dantzig-Wolfe Decomposition

- A common use of column generation is to separate a set of “easy” constraints from the rest of a large LP. This is called Dantzig-Wolfe decomposition.
- There are many reasons why the “easy” constraints may be easier than the full formulation:
  - The easy constraints may involve only a small fraction of the variables.
  - The easy constraints may have a special structure, such as a graph algorithm, which can be solved quickly using a special algorithm.

Dantzig-Wolfe Decomposition

- A common use of column generation is to separate a set of "easy" constraints from the rest of a large LP. This is called Dantzig-Wolfe decomposition.
- There are many reasons why the "easy" constraints may be easier than the full formulation:
  - The easy constraints may involve only a small fraction of the variables.
  - The easy constraints may have a special structure, such as a graph algorithm, which can be solved quickly using a special algorithm.
- The idea is to solve the easy constraints separately (and repeatedly, as it turns out).

Extreme points and extreme rays

Consider a set of linear inequalities: $\sum_j a_{ij} x_j \leq b_i$. The feasible region for this system has a set of extreme points $x_1^*, \ldots, x_s^*$ and extreme rays $d_1^*, \ldots, d_t^*$.
Theorem: Any point in the feasible region defined by the set of linear constraints above may be expressed in the form

$$\sum_{i=1}^{s} \alpha_i x_i^* + \sum_{j=1}^{t} \beta_j d_j^*,$$

where $\alpha_i, \beta_j \geq 0$ and $\sum_{i=1}^{s} \alpha_i = 1$.

Extreme points and extreme rays

Consider a set of linear inequalities: $\sum_j a_{ij} x_j \leq b_i$. The feasible region for this system has a set of extreme points $x_1^*, \ldots, x_s^*$ and extreme rays $d_1^*, \ldots, d_t^*$.

Theorem: Any point in the feasible region defined by the set of linear constraints above may be expressed in the form

$$\sum_{i=1}^{s} \alpha_i x_i^* + \sum_{j=1}^{t} \beta_j d_j^*,$$

where $\alpha_i, \beta_j \geq 0$ and $\sum_{i=1}^{s} \alpha_i = 1$.

The main idea of Dantzig-Wolfe decomposition is to replace the system of inequalities with this expression and generate the columns x_i^* or d_j^* as needed using column generation.