Cutting plane techniques for getting improved bounds
Quick Review of Branch and Bound

<table>
<thead>
<tr>
<th>Prize</th>
<th>iPod</th>
<th>Server</th>
<th>brass rat</th>
<th>gift certificate</th>
<th>6.041</th>
<th>15.053</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Utility</td>
<td>16</td>
<td>22</td>
<td>12</td>
<td>8</td>
<td>11</td>
<td>19</td>
</tr>
</tbody>
</table>

Budget: 14 IHTFP points.

maximize $16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6$
subject to $5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14$
x$_j$ binary for $j = 1$ to $6$

AS you will see, it is a very quick review.
The only thing I wanted to focus on was that we fathom a node j of the branch and bound tree whenever $z_{LP}(j) \leq z_i$. 

Explore subproblems one at a time. Fathom node j if $z_{LP}(j) \leq z_i$. 

Branch and Bound 

$z_i = 43$
On Improving B&B Through Better Bounds

Explore subproblems one at a time. Fathom node j if $z_{LP}(j) \leq z_i$.

Way to speed up things: fathom nodes sooner!

Strategy 1: Improve $z_i$. This means seeking out better solutions either at the beginning of B&B or during it.

Strategy 2: Improve $z_{LP}(j)$. This means coming up with better bounds.

What is a better bound?  
How does one get it?

The branch and bound tree is huge. With 100 binary variables, it is of size $2^{100}$. To give you a sense for how big that is, the universe has only existed for less than $2^{60}$ seconds.

The only chance for branch and bound to work with 100 binary variables is to fathom nodes that are not far from the root node (that is node 1) of the tree. Improving the quality of an incumbent is very important. But here, we will focus on improving the bound.
On Bounds from Linear Programming

Better Bounds means bounds that are closer to the true optimum integer value.

For maximization problems we are computing upper bounds. But we want the upper bound to be as low as possible.

44 \(< z_{LP}(1) < 45. We cannot fathom node 1, even if we start with the optimum solution as the incumbent.

NEXT: an IP formulation s.t. \( z_{LP}(1) < 44. \)

We could fathom node 1, if \( z_i = 43. \)

We will next show that we can easily improve the bound for the IHTFP game show problem. Moreover, the branch and bound algorithm can possibly terminate after looking only at node 1 of the tree.
<table>
<thead>
<tr>
<th>Prize</th>
<th>iPod</th>
<th>server</th>
<th>brass rat</th>
<th>gift certificate</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Budget: 14 IHTFP points.

Claim: Nooz can select at most two of prizes 1, 2, 3 and 6. (Check it out).

Therefore: we can add the constraint

\[ x_1 + x_2 + x_3 + x_6 \leq 2 \]

to the original IP and it is still a correct IP model. It has the same set of integer solutions as before.

We note that the number of points needed for prizes 1, 2, 3 and 6 are such that at most two of the prizes can be selected. Any three of the prizes would use up at least 15 IHTFP points.

So, we can add the constraint \( x_1 + x_2 + x_3 + x_6 \leq 2 \). This new constraint does not eliminate any feasible integer solutions, and so we can add it to the integer program. But it does change the LP relaxation of the integer program.
A Revised Integer Program

maximize \(16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6\)

subject to \(5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14\)
\(x_1 + x_2 + x_3 + x_6 \leq 2\)
\(x_j \text{ binary for } j = 1 \text{ to } 6\)

\(z_{IP} = 43; \quad z_{LP} = 43\ 3/4\)

Suppose \(z_i = 43\)

Fact: improving formulations of IPs to get better bounds has been critical to solving larger IPs faster.

When we used Excel to find the optimum solution to LP(1), we obtained an objective value of \(z_{LP}(1) = 43.75\). This means that Bound(1) = 43.

We will also assume that we somehow discovered the integer solution with objective value 43, and that it was our current incumbent. (There is no guaranteed way of finding such a good solution. But often we can find such good solutions, and the major purpose in branch and bound is to prove that there are no better solutions.)

In any case, if Bound(1) = 43, and if \(z_i = 43\), we can fathom node 1. At this point, there are no active nodes, and the algorithm terminates.

We won’t always be so fortunate as to be able to solve integer programs in a single bound, but we often can solve integer programs with a relatively small number of branches.
Valid Constraints and Better Bounds

- We say a constraint is \textit{valid} for an integer program if it is satisfied by all feasible solutions.
  - The constraint $x_1 + x_2 + x_3 + x_6 \leq 2$ was valid.

- How can we find valid constraints?
  - There are lots of techniques.
  - We will show you Gomory cuts later
  - But it is also very challenging to come up with valid constraints.
  - And we want constraints that lead to improved bounds for the LP relaxation. This is even more challenging.

---

Here we want to emphasize that we found a better bound by adding a valid constraint to the integer program. This is a standard way of getting better bounds. In fact, it is often referred to as getting improved IP formulations, meaning that the LP relaxation gives better bounds.
On Page 656 of The American Mathematical Monthly, Vol. 83, No. 8. (Oct., 1976), Sidney Penner from the Bronx Community College, CUNY, proposed the following elementary problem E2612:

"How many diamonds can be packed in a Chinese checkerboard? This board consists of 2 order 13 triangular arrays of holes, overlapping in an order 5 hexagon, 121 holes in all. A diamond consists of four marbles that fill four adjacent holes."

On Pages 199 and 200 of The American Mathematical Monthly, Vol. 85, No. 3. (Mar., 1978) , a solution by Paul Vojta, a student at the University of Minnesota, was reported. Others had submitted correct solutions as well (including the proposer).

The connection to integer programming was established by Vasek Chvátal in his paper on "Cutting planes in combinatorics," European J.Combin. 6 (1985), 217-226. In particular, he gives a cutting-plane proof of optimality, and he also exhibits a dual solution.
What is the maximum number of diamonds that can be packed on a Chinese checkerboard.

Each diamond covers 4 dots.

It turns out that it is not too difficult to come up with the best packing. But it is very tricky to prove that it is the best packing.
The diamonds are not permitted to overlap, or even to share a single circle.

Here is a pretty good, but not optimal packing.
What is the best packing you can find?

Try it out if you have not already done so.
In slide show mode, the best packing is revealed. It has 27 diamonds.
Set Packing Problem

- Let \( D \) be the collection of diamonds
- Let \( O \) be the pairs of diamonds that overlap.
  - \((d, d') \in O\), implies that diamonds \( d \) and \( d'\) have at least one point in common

- Decision variables: \( x_d \) for \( d \in D \)
  - \( x_d = 1 \) if diamond \( d \) is selected
  - \( x_d = 0 \) if diamond \( d \) is not selected.

We will try to use integer programming to come up with the best packing. Hopefully, we will also come up with a proof that it is the best packing.

One natural way of expressing the constraints is to say that two overlapping diamonds can both be selected. We shall turn this into an integer programming constraint in two slides.
How many decision variables are there?

There are 88 black circles. For each black circle, one can hang a yellow diamond from it. So, there are 88 possible yellow diamonds.

And there are 88 green and red diamonds. for a total of 264 diamonds.

Here we point out that there is one variable for each possible diamond, and the total number of diamonds is 264.
The First Integer Programming Formulation

Maximize \[ \sum_{d \in D} x_d \]
subject to \[ x_d + x_{d'} \leq 1 \] for all \((d, d') \in O\)
\[ 0 \leq x_d \leq 1 \] and \(x_d\) integer for \(d \in D\)

This problem is known as the set packing problem.

\[ z_{IP} = 27 \]
\[ z_{LP} = 132 \]

The LP opt has \(x_d = 1/2\) for each diamond \(d\).

This would be terrible for B&B!

There are many more constraints, but there are fewer than 3000 constraints. Perhaps we could use integer programming branch and bound. But we can’t!

The LP relaxation can be solved by setting each variable to \(1/2\). This gives an objective value of 132. So, we conclude that no feasible solution to the packing problem has more than 132 diamonds.

But this bound is horrible. In fact, if you notice that there are only 121 dots on a Chinese Checkerboard, you would figure out that the maximum number of diamonds is at most \(121/4 = 30.25\).

So, is the linear programming bound so horrible? It’s because it is not a very good formulation of the integer program if we care about good bounds.
For each circle, you can select at most one diamond incident to the circle.

Here we come up with an alternative formulations, with one constraint per dot of the Chinese Checkerboard. It is a far better formulation.

Here we express the constraint as follows: there is at most one diamond that covers any specified dot of the checkerboard.
An improved integer program

Let $S(i)$ be the diamonds that are incident with circle $i$ of the Chinese Checkerboard.

$$z_{IP} = \text{Max} \quad \sum_{d \in D} x_d$$

s.t. \quad \sum_{d \in S(i)} x_d \leq 1 \quad \text{for each circle } i

\quad x_d \in \{0,1\} \quad \text{for all } d \in D

Let $S(i)$ be the diamonds that are incident with circle $i$ of the Chinese Checkerboard.

If we express the integer program with these 121 constraints, we find that $z_{LP}(1) = 27.5$. Therefore, $\text{Bound}(1) = 27$. Since we have already found a solution that packs 27 diamonds, the Branch and Bound algorithm terminates.

So, with the poor (but easily understood and correct) formulation of the integer program, we could never solve the problem using Branch and Bound. The bounds would be so poor that we would have to enumerate most of the tree.

But with the improved constraints, and thus an improved IP, Branch and Bound terminated after a single bound.

Next we will look at some geometry to get a sense where the valid constraints are coming from and how to construct them.
Example. maximize \(-x - 10y\)
subject to \(x, y \text{ are in } P\)
\(x, y \text{ integer}\)

Here \(P\) is the yellow region, and the feasible solutions are the black dots in \(P\). The optimal fractional solution is the red dot at the bottom of the feasible region. But the best integer solution is far away.
Valid Inequalities and Cutting Planes

Any additional constraint that does not eliminate a feasible integer point is called a **valid inequality**.

A **cut** is a valid inequality that eliminates some of the LP feasible region.

But the yellow region is not the “best feasible region” for the IP. In fact, we would like to add constraints that cut off parts of the yellow region without cutting off any feasible integer points.

A constraint that does not eliminate any feasible integer point is a valid inequality. (I suppose we could develop valid equalities as well, but techniques always develop valid inequalities.)

The best kind of valid inequality is one that eliminates part of the LP feasible region, that is, it cuts off part of the yellow region. We call such a valid inequality a **cut**. We are using cut here very differently than we did when discussing the max flow algorithm. The two types of cuts have nothing to do with each other.
On adding cuts

For a max problem, we want to add cuts to make $z_{LP} - z_{IP}$ smaller.

$z_{LP} - z_{IP}$ is reduced when $z_{LP}$ is reduced.

In this case, we added a cut, but $Z_{LP}$ did not change.

Adding a cut is good because it has a chance of making $z_{LP}$ lower since part of the feasible region is cut off. And a lower value of $z_{LP}$ means a better bound.
The Best LP feasible region for an IP

The convex hull of X is the smallest LP feasible region that contains X.

Every corner point of the convex hull of X is integer.

If we solve the linear program over the convex hull of X, we get an integer solution.

The best valid inequalities would essentially shrink wrap the feasible region. It would create the smallest feasible region containing all of the integer solutions. We illustrate this in blue-green in the diagram.

The name for the smallest convex set containing X is the convex hull of X.

It’s interesting that every corner point of ConvexHull(X) is a point of X. So, if we were to optimize over the LP relaxation, we would get a corner point (LPs using the simplex algorithm always give corner point solutions), and this would be a point of X.

Put another way, optimizing over the Convex Hull of X will also give the optimum solution over X.
This is just a cute animation showing the feasible region of an LP collapsing into the convex hull.
How Cutting Plane Algorithms Work

1. Solve the LP relaxation of an IP
2. If you can find a valid cut that will reduce the LP bound, then do it.
3. Iterate until you have the best bound you can get.

FACTS
- Cutting plane approaches dramatically speed up B&B through better bounding.
- There is a lot of research on how to find cuts that work well in practice.
Mental Break

Another top 10 list
Actually, Gomory Cuts was designed to be more than a cut generator. It was designed to be in and of itself and IP solver. The idea was to develop cuts one at a time until eventually, the LP obtained an integer optimal solution. It was not designed as part of Branch and Bound.

Theoretically, Gomory cuts can be used to solve an integer program, and they are guaranteed to work in “finite time.”

Unfortunately, they solved integer programs very slowly. So, they were largely ignored during the 1970s and 1980s and 1990s. Only recently was it discovered that this same approach was very effective is used with Branch and Bound.
Gomory Cuts

- **Step 1.** Solve the linear programming relaxation to optimality
- **Step 2.** If the optimum solution to the LP is integer valued, then quit with an optimum solution for the IP. Otherwise, find a constraint in the final tableau in which the basic variable is fractional.
- **Step 3.** Use the constraint found in Step 2 to create a valid inequality called a Gomory Cut. The current LP optimum solution is eliminated by the cut. Solve the new LP again using the Dual Simplex Algorithm.

The idea of generating cuts automatically is a natural one since we want integer programming solvers to find the optimum solution without lots of human involvement.

In order to accomplish this automatic generation of cuts, we start from the final tableau of the LP relaxation. If the optimal solution to an LP relaxation is integer valued, then it is optimal for the Integer program, and we are done. Otherwise, we can take a constraint of the final tableau, create a Gomory cut, add the cut to the current LP relaxation, and get a new and improved LP relaxation.

We solve the new relaxation using the dual simplex method since this method can create a great starting point by using the optimal tableau for the previous LP relaxation.
On fractions and inequalities

Example 1. \[1.6 x - y = 1.8\]
\[x \geq 0, \quad y \geq 0 \quad x, y \text{ integer}\]

It follows that \[.6x \geq .8\] why?

The fractional part of \(1.6x - y\) is \(.8\).

Same as the fractional part of \(.6x\)

Also, \(.6x \geq \) fractional part of \(.6x\)

Example: consider \(x = 3\) and \(y = 3\).
\[.6x = 1.8 > \text{fractional part of} \ .6x \ = .8.\]

Fractional Parts
\[\text{fr}(1.3) = .3\]
\[\text{fr}(2.1) = .1\]
\[\text{fr}(2) = 0\]
\[\text{fr}(-.3) = .7\]
\[\text{fr}(-2.6) = .4\]
\[0 \leq \text{fr}(a) < 1.\]

We will soon show how to take an equality constraint from the final tableau and create a Gomory cut. The approach relies on some elementary properties of fractions and fractional parts.

First note how fractional parts are defined by looking at the right part of the slide. It is always a real number that is at least 0 and less than 1.

Example one shows how to obtain a valid inequality from an equality constraint. If we know that \(1.6 x - y = 1.8\) and both \(x\) and \(y\) are non-negative, then it follows directly that \(.6x \geq .8\)  (There are other constraints that are also implied, but this particular constraint is easily derived and easily explained.)

To obtain the constraint, all one has to do is focus on fractions. Somehow, we need the fractional part of \(1.6 x - y\) to be \(.8\). But the fractional part of \(1.6x - y\) is the fractional part of \(.6 x\). Does this mean that \(.6x = .8\)? No, it is possible that \(.6x = 1.8\) for example. Rather, we know that \(.6x\) is \(.8\) plus an integer. So, it immediately follows that \(.6x \geq .8\).
More on working with fractional parts

Example 2. \(2.3 \times x - 0.6 \times y = 3.9\)
\[x \geq 0, \quad y \geq 0 \quad x, y \text{ integer}\]

It follows that \(0.3 \times x + 0.4 \times y \geq 0.9\) why?

Fractional part of \(2.3 \times x - 0.6 \times y\) is the same as the fractional part of \(0.3 \times x + 0.4 \times y\).
This equals the fractional part of 3.9, which is 0.9

\[0.3 \times x + 0.4 \times y \geq \text{FractionalPart} (0.3 \times x + 0.4 \times y) = 0.9\]

\[	ext{e.g., } x = 3, \quad y = 5 \quad 0.3 \times x + 0.4 \times y = 2.9\]

On this slide, we consider a more general equality, with two components with fractional values.

As before, we know that the fractional part of \(2.3 \times x - 0.6 \times y\) is 0.9.
This means that the fractional part of \(0.3 \times x + 0.4 \times y\) is 0.9
And as before, it means that \(0.3x + 0.4y = 0.9\) plus an integer.
So, we conclude that \(0.3x + 0.4y \geq 0.9\).

It may seem that we can come up with an even better inequality. Perhaps. But the inequality \(0.3x + 0.4y\) is just fine for our purposes.
We now formalize the approach for developing an inequality from an equality constraint. This approach works whenever the RHS is non integer. It is the approach that was used by Gomory to develop his cuts.
Zor’s LP again, with a small change

<table>
<thead>
<tr>
<th>z</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-52</td>
<td>-30</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>200</td>
</tr>
</tbody>
</table>

Final Tableau

<table>
<thead>
<tr>
<th>z</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>4</td>
<td>1160</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1/8</td>
<td>0</td>
<td>1</td>
<td>-5 3/4</td>
<td>3/8</td>
<td>2 1/2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5/8</td>
<td>1</td>
<td>0</td>
<td>1 1/4</td>
<td>-1/8</td>
<td>12 1/2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3/8</td>
<td>0</td>
<td>0</td>
<td>-1/4</td>
<td>1/8</td>
<td>17 1/2</td>
</tr>
</tbody>
</table>

We now illustrate finding Gomory cuts on an integer program.

We return to Zor’s problem. Actually, the original problem discussed in the duality lecture had an integer optimum solution. So, we modified the RHS a little so that the optimum solution to the LP relaxation would be fractional.
Finding a cut.

<table>
<thead>
<tr>
<th></th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>4</td>
<td>1160</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1/8</td>
<td>0</td>
<td>1</td>
<td>-5 3/4</td>
<td>3/8</td>
<td>2 1/2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5/8</td>
<td>1</td>
<td>0</td>
<td>1 1/4</td>
<td>-1/8</td>
<td>12 1/2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3/8</td>
<td>0</td>
<td>0</td>
<td>-1/4</td>
<td>1/8</td>
<td>17 1/2</td>
</tr>
</tbody>
</table>

We assume here that we Zor’s problem is an IP and we have solved the LP relaxation and obtained the optimum tableau.

Select any constraint from the LP final tableau that has a fractional RHS. Use this to create a cut.

There are three constraints involving the basic variables as well as the z-row. We can use the z-row for cuts as well if the RHS is non-integer and if z is required to be integer valued or if all variables are required to be integer and all cost coefficients are integer.

So, we could create three different Gomory cuts. We will look at the first constraint only and create a Gomory cut.
Creating a Gomory Cut

<table>
<thead>
<tr>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1/8</td>
<td>0</td>
<td>1</td>
<td>-5 3/4</td>
<td>3/8</td>
<td>2 1/2</td>
</tr>
</tbody>
</table>

\[ \frac{1}{8} x_2 + s_1 - 5 \frac{3}{4} s_2 + 3/8 s_3 = 2 \frac{1}{2}. \]

Replace all coefficients by fractional parts.
Replace the equality by a “≥”

\[ \frac{1}{8} x_2 + \frac{1}{4} s_2 + 3/8 s_3 \geq \frac{1}{2} \]

We carry out the rule defined just a few slides ago, and we derive the constraint \( \frac{1}{8} x_2 + \frac{1}{4} s_2 + 3/8 s_3 \geq \frac{1}{2} \), which is a Gomory cut.
Once a Gomory cut is created, it is added to the previous LP relaxation creating an improved LP relaxation. Since the Gomory cut is an inequality constraint, we also need to add a surplus variable $s_4$ in order to create a basic solution.

Here we create a basic solution with variables $x_1$, $x_3$, $s_1$, and $s_4$. The basic solution is dual feasible, and so one can use the Dual Simplex Algorithm to find an optimal feasible solution.
After 1 dual pivot

<table>
<thead>
<tr>
<th></th>
<th>z</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>s₄</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1 1/2</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>2.5</td>
<td>4</td>
<td></td>
<td>1158</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2/3</td>
<td>1</td>
<td>0</td>
<td>1 1/3</td>
<td>0</td>
<td>-1/3</td>
<td>12 2/3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>-1/3</td>
<td>0</td>
<td>1/3</td>
<td>17 1/3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>2/3</td>
<td>1</td>
<td>-2 2/3</td>
<td>1 1/3</td>
<td></td>
</tr>
</tbody>
</table>

Note. The LP bound for this tableau is better (lower) than that for the last tableau.

There are three fractional constraints that we can use to create the next Gomory cut. (Generating several at a time is a good approach.)

With only one additional pivot, we have found an optimal solution for this new LP relaxation. Moreover, the objective value has decreased from 1160 to 1158, and thus the upper bound is better. (It is closer to the true integer solution.)

Unfortunately, we are still not optimal since we do not have an integer solution. There are three constraints with fractional right hand sides. So, we can create three new Gomory Cuts if we wish.
Here we leave it as an exercise to create two more Gomory Cuts.
On Gomory Cuts

Gomory cuts is a very nice way of generating cuts for improving bounds in B&B

- easy to automate
- the current optimum LP solution is cut off by the Gomory cut

One can solve the IP by just adding Gomory cuts, but this is a very slow procedure.

As mentioned earlier, Gomory cuts can be a great way of improving an LP relaxation. But it is not a great way to solve an LP in and of itself. It really should be used as part of a Branch and Bound procedure to be effective.
Problem Specific Cuts

- For many combinatorial problems there are special purpose cuts that yield better bounds.
  - Knapsack
  - 053 chocolates (covering problems)
  - Traveling Salesman Problem
  - many more

We won’t discuss it here, but researchers have developed lots of methods of finding cuts for specific problems. So, there are special routines that can find cuts for knapsack problems. And there are special routines for finding cuts for the traveling salesman problem. These techniques are useful, but there is not time to develop them in this lecture.
A Return to Diamond Packing

We will show the bound of 27.5 by looking at the dual.

This doesn’t really have anything to do with “cuts”, but it’s cool.

In slide show mode, the best packing is revealed. It has 27 diamonds.
A return to Diamond Packing

Assign each circle (or node) a “weight.”

The weight of a diamond $d$ is $w(d) = \text{the sum of its node weights}$.

The dual linear program: choose weights for each circle so that $w(d) \geq 1$ for each diamond $d$. Minimize the weights of the circles.

This really has nothing to do with cuts. What we will show soon is how to visualize a solution to the dual problem of Diamond Packing so that one can see why there is at most 27.5 diamonds that can be packed.

The dual problem assigns dual variables to constraints.
Each constraint corresponds to a circle of the Chinese Checkerboard.
So, the dual problem assigns a dual variable (or weight) to each circle.

There is a constraint in the dual for each variable of the primal problem, and thus for each diamond.
The dual constraint says that the weight of the circles that a diamond covers is at least 1.
Each diamond $d$ has weight $w(d) \geq 1$. Example, assign all circles a weight of $\frac{1}{4}$.

There are 121 circles or nodes.

Number of Diamonds $\leq \sum_d w(d) \leq \sum_j w_j$

$\sum_j w_j = \frac{121}{4} = 30.25$

That is, the number of diamonds is at most 30.

Here we develop a feasible solution to the dual problem. Assign each circle a weight of $\frac{1}{4}$. Then each diamond has a weight of exactly 1. Since there are 121 circles, the objective for the dual problem is 30.25, which is the weight of the circles.

So, no solution has more than 30.25 diamonds.

Incidentally, one naturally sees the bound of 30.25 by recognizing that each diamond includes 4 circles, and thus the number of diamonds is at most the number of circles divided by 4.
Each diamond has weight at least 1.

Number of Diamonds \( \leq \sum_d w(d) \leq \sum_j w_j \)

\[ \sum_j w_j = \frac{43}{4} + 18 = 28.75 \]

That is, the number of diamonds is at most 28.

But one can develop even better bounds. Here we have three types of weights. There are the small black circles with a weight of 0, the blue inner circles with a weight of \( \frac{1}{4} \), and the large red circles with a weight of 1. You can see that the weight of the circles covered by any diamond is at least 1.

The total weight of the circles is 28.75 and thus there are at most 28 diamonds that can be packed. But we can do better.
In this example, every diamond has a weight of at least 1. And the sum of the weights is 27.5. Thus there are at most 27 diamonds that can be packed.
The Optimal Solution Again
What makes a really good integer programming formulation? Assuming that one is going to use Branch and Bound, we want a formulation such that the LP relaxation gives low bounds (assuming a maximization problem). A good IP formulation leads to an improved solution time.

As for Integer Programming, it’s incredibly useful in practice because of its ability to model logical constraints, nonlinearities, as well as natural situations in which solutions need to be integer valued. It’s not as easy to model, and it’s much harder to solve. But it’s worth the effort when it can be solved.
IP Solution Techniques Summary

- **Branch and Bound**
  - very general and adaptive
  - used in practice (e.g. Excel Solver)

- **Cutting planes**
  - clever way of improving bounding
  - active area for research, theoretical and applied