Solving Integer Programs with Branch and Bound (and Branch and Cut)

Cynthia Phillips (Sandia National Laboratories)

Reconnect ‘04

One more general branch and bound point

Node selection: when working in serial, how do you pick an active node to process next?

Usual: best first
- Select the node with the lowest lower bound
- With the current incumbent and solution tolerances, you will have to evaluate it anyway
- If you don’t have a good incumbent finder, you might start with diving:
  - Select the most refined node
  - Once you have an incumbent, switch to best first

Mixed-Integer programming (IP)

\[
\begin{align*}
\text{Min} & \quad c^T x \\
\text{Subject to:} & \quad Ax = b \\
& \quad \ell \leq x \leq u \\
& \quad x = (x_I, x_C) \\
& \quad x_I \in \mathbb{Z}^n \quad \text{(integer values)} \\
& \quad x_C \in \mathbb{Q}^n \quad \text{(rational values)}
\end{align*}
\]

Branch and Bound

\[\text{Bounding function: solve LP relaxation}\]
\[\text{Branching:}\]
- Select an integer variable \(x_i\) that’s fractional in the LP solution \(x^*\)
  - Up child: set \(x_i = \lceil x_i^* \rceil\)
  - Down child: set \(x_i = \lfloor x_i^* \rfloor\)
- \(x^*\) is no longer feasible in either child

\[\text{Incumbent method:}\]
- many incumbent-finding methods to follow in later lectures
- \(\text{Obvious: return the LP solution if it is integer}\)
- \(\text{Satisfies feasibility-tester constraint}\)
Solving Integer Programs: Branch and Bound

Root Problem = original

- \( x_j = 0 \)
- \( x_j = 1 \)

fathomed

infeasible

\* Lower bound: LP relaxation

Branching Decisions: One method

Which candidate variable (of thousands or more choices) to choose?
- Use structural insight (user priorities) when possible.
- Use gradients (pseudocosts):

\[
\begin{align*}
\text{Down: } & \frac{15 - 12}{0.3} = 10 \\
\text{Up: } & \frac{13.4 - 12}{1 - 0.3} = 2
\end{align*}
\]

Using gradients

- Use gradients to compute an expected bound movement for each child
  - \( \text{Up} = \text{up gradient} \times \text{upward round distance} \)
  - \( \text{Down} = \text{down gradient} \times \text{downward round distance} \)
- Try to find a variable for which both children might move the bound

Initialization

- When a variable is fractional for the first time, initialize by “pretending” to branch (better than using an average)

Branch Variable Selection method 2: Strong Branching

- “Try out” interesting subproblems for a while
  - Could be like a gradient “initialization” at every node
  - Could compute part of the subtree
- This can be expensive, but sometimes these decisions make a huge difference (especially very early)
**Branching on constraints**

Can have one child with new constraint: \( a^T x \leq b \)
and the other with new constraint: \( a^T x \geq b \)
- Must cover the subregion
- Pieces can be omitted, but only if provably have nothing potentially optimal
- Node bounds are just a special case

More generally, partition into many children
\[
\begin{align*}
  a^T x &\leq b_1, \quad b_2 \leq a^T x &\leq b_3, \quad \ldots, \quad b_{k-1} \leq a^T x &\leq b_k
\end{align*}
\]

---

**Special Ordered Sets (SOS)**

- Models selection: \[
\sum_{i \in I} y_i = 1
\]
\( y_i \in \{0,1\} \)

Example: time-indexed scheduling
- \( x_j = 1 \) if job \( j \) is scheduled at time \( t \)
- Ordered by time

---

**Special Ordered Sets: Restricted Set of Values (Review)**

- Variable \( x \) can take on only values in \( \{v_1, v_2, \ldots, v_m\} \)
  - Frequently the \( y_i \) are sorted
  - Capacity of an airplane assigned to a flight

\[
\begin{align*}
  x &= \sum_{i=1}^{m} y_i v_i \\
  \sum_{i=1}^{m} y_i &= 1 \\
  y_i &\in \{0,1\}
\end{align*}
\]

- The \( y_i \)'s are a special ordered set.

---

**Problems with Simple Branching on SOS**

Generally want both children to differ from parent substantially
For SOS
- The up child (setting a variable to 1) is very powerful
  - All others in the set go to zero
- The down child will likely have an LP solution = parent’s
  - “Schedule at any of these 1000 times except this one”
Special Ordered Sets: Weak Down-Child Example

- Variable \( x \) can take on only values in \( \{v_0, v_1, \ldots, v_n\} \)
- Plane capacities, values \( \{50, 100, 200\} \)

\[
x = \sum_{i=1}^{n} y_i v_i, \\
\sum_{i=1}^{n} y_i = 1, \\
y \in \{0, 1\}
\]

If \( v_2 = 0 \) (can’t use capacity 100), “fake” a 100-passenger plane by using \( \frac{2}{3} x_1 + \frac{1}{3} x_3 \).

Branching on SOS

- Set variables \( \{v_0, v_1, \ldots, v_n\} \)
- Partition about an index \( i \)

- Up child has \( \sum_{j=1}^{i} x_j = 1 \)
- Down child has \( \sum_{j=i+1}^{n} x_j = 1 \)

- Examples:
  - Schedule job \( j \) before/after time \( t \)
  - Use a plane of capacity at least/at most \( s \)

Branching causes exponential growth of the search tree

Is there a way to make progress without branching?
Strengthen Linear Program with Cutting Planes

- Make LP polytope closer to integer polytope
- Use families of constraints too large to explicitly list
  - Exponential, pseudopolynomial, polynomial ($n^4$, $n^5$)

Example: Maximum Independent Set

\[
v_i = \begin{cases} 
1 & \text{if vertex } i \text{ is in the MIS} \\
0 & \text{otherwise}
\end{cases}
\]

\[
\max \sum v_i \\
s.t. v_i + v_j \leq 1 \quad \forall (i, j) \in E
\]

Example: Some Maximum Independent Set Cutting Planes

\[
v_i + v_j + v_k \leq 1 \quad \forall \text{triangles } i, j, k
\]

More general clique inequalities (keep going till they’re too hard to find)

Value of a Good Feasible Solution Found Early

- Simulate seeding with optimal (computation is a proof of optimality)
- Note: proof of optimality depends on a good LP relaxation
Solving Subproblem LPs Quickly

- The LP relaxation of a child is usually closely related to its parent’s LP
- Exploit that similarity by saving the parent’s basis

Linear Programming Geometry

Optimal point of an LP is at a corner (assuming bounded)

\[ c^T x = \text{opt} \]

\[ c^T x = -\infty \]

Linear Programming Algebra

What does a corner look like algebraically?

\[ Ax = b \]

Partition A matrix into three parts

\[
\begin{pmatrix}
B & L & U
\end{pmatrix}
\]

where B is nonsingular (invertible, square).

Reorder \( x = (x_B, x_L, x_U) \)

We have \( Bx_B + Lx_L + Ux_U = b \)

Linear Programming Algebra

We have \( Bx_B + Lx_L + Ux_U = b \)

Set all members of \( x_L \) to their lower bound.

Set all members of \( x_U \) to their upper bound.

Let \( B' = b - Lx_L - Ux_U \) (this is a constant because bounds \( l \) and \( u \) are)

Thus we have \( Bx_B = b' \)

Set \( x_B = B^{-1}b' \)

This setting of \( (x_B, x_L, x_U) \) is called a basic solution

• A basic solution satisfies \( Ax = b \) by construction

If all \( x_B \) satisfy their bounds \( (l \leq x_B \leq u) \), this is a basic feasible solution (BFS)

Slide 21

Slide 22

Slide 23

Slide 24
Basic Feasible Solutions

We have $Bx + Lx + Ux = b$
Set all members of $x_i$ to their lower bound.
Set all members of $x_i$ to their upper bound.
Let $b' = b - Lx - Ux$ (this is a constant because bounds $l$ and $u$ are)
Set $x = b'x$

In the common case $l = 0$, $u = (x \geq 0)$,
we have $b = b'$, $x_L = \emptyset$, $x_U = N$
x are basic variables, $N$ are nonbasic ($x_L$ are nonbasic at lower, $x_U$ are nonbasic at upper)

Algebra and Geometry

A BFS corresponds to a corner of the feasible polytope:
$Ax \leq b$
$m$ inequality constraints (plus bounds) and $n$ variables. Polytope in $n$-dimensional space. A corner has $n$ tight constraints.

With slacks $Ax + x_L = b$
$m$ equality constraints (plus bounds) in $n+m$ variables. A BFS has $n$ tight bound constraints (from the nonbasic variables).

Dual

(Simplified) primal LP problem is
minimize $c^T x$
such that $Ax \leq b$
$x \geq 0$

The dual problem is:
maximize $y^T b$
such that $yA \geq c$
y $\geq 0$
• dual(dual(primal)) = primal
• Frequently has a nice interpretation (max flow/min cut)

LP Primal/Dual Pair

Primal feasible
Dual feasible

Opt
Parent/Child relationship (intuition)

**Parent optimal pair (x*,y*)**
- Branching reduces the feasible region of the child LP with respect to its parent and increases the dual feasible region
- y* is feasible in the child’s dual LP and it’s close to optimal

Resolving the children using dual simplex, starting from the parent’s optimal basis can be at least an order of magnitude faster than starting from nothing.

Note: Basis can be big. Same space issues as with knapsack example.