Reconnect '04
Solving Integer Programs with Branch and Bound (and Branch and Cut)

Cynthia Phillips (Sandia National Laboratories)

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)

Min $c^T x$

Subject to: $Ax = b$

$\ell \le x \le u$

$x = (x_I, x_C)$

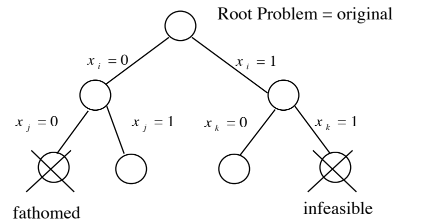
$x_I \in Z^n$ (integer values)

$x_C \in Q^n$ (rational values)

Branch and Bound

- Bounding function: solve LP relaxation
- Branching
  - Select an integer variable x_i that's fractional in the LP solution x^*
  - Up child: set $x_i \ge \lceil x_i^* \rceil$
  - Down child: set $x_i \le \lfloor x_i^* \rfloor$
  - x^* is no longer feasible in either child
- Incumbent method
  - many incumbent-finding methods to follow in later lectures
  - Obvious: return the LP solution if it is integer
    - Satisfies feasibility-tester constraint

Solving Integer Programs: Branch and Bound

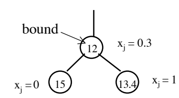


- Lower bound: LP relaxation

Branching Decisions: One method

Which candidate variable (of thousands or more choices) to choose?

- Use structural insight (user priorities) when possible.
- Use gradients (pseudocosts):



Down: $\frac{15-12}{0.3} = 10$ Up $= \frac{13.4-12}{1-0.3} = 2$

Using gradients

- Use gradients to compute an expected bound movement for each child
  - Up = up gradient * upward round distance
  - Down = down gradient * 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, intialize 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: $a^T x \le b_1$
and the other with new constraint: $a^T x \ge b_2$
• 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

$$a^T x \le b_1,\ \mathrm{b}_2 \le a^T x \le b_3,\ \ldots,\ b_{k-1} \le x \le b_k$$

Special Ordered Sets (SOS)

• Models selection: $\sum_{i=1}^{m} y_i = 1$
$y_i \in \{0,1\}$

Example: time-indexed scheduling
• $x_{jt} = 1$ 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 v_i are sorted
    • Capacity of an airplane assigned to a flight

$$x = \sum_{i=1}^{m} v_i y_i$$
$$\sum_{i=1}^{m} y_i = 1$$
$$y \in \{0,1\}$$

  – 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 $\{v_1, v_2, \dots v_m\}$
- Plane capacities, values {50,100,200}

$$x = \sum_{i=1}^{m} v_i y_i$$
$$\sum_{i=1}^{m} y_i = 1$$
$$y \in \{0,1\}$$

If v2=0 (can't use capacity 100), "fake" a 100-passenger plane by using

$$\frac{2}{3}x_1 + \frac{1}{3}x_3$$

Sandia National Laboratories

Branching on SOS

Set variables $\{x_1, x_2, \dots x_m\}$ $\sum_{j=1}^{m} x_j = 1$

Partition about an index *i:*

- Up child has $\sum_{j=i+1}^{m} x_j = 1$
- Down child has $\sum_{j=1}^{i} x_j = 1$
- Examples:
  - Schedule job *j* before/after time *t*
  - Use a plane of capacity at least/at most *s*

Sandia National Laboratories

Branching on SOS

- Good choice for partition point *i* is such that as nearly as possible

$$\sum_{j=1}^{i} x_j = \sum_{j=i+1}^{m} x_j = \frac{1}{2}$$

- Can compute gradients as with simple variable branching
- This is **not** general branching on a constraint: just setting multiple variable upper bounds simultaneously

Sandia National Laboratories

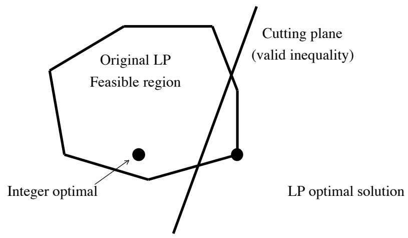
Postponing Branching

Branching causes exponential growth of the search tree

Is there a way to make progress without branching?

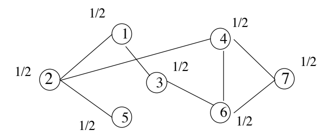
Sandia National Laboratories

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

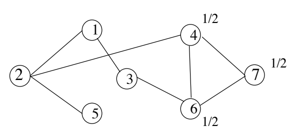


$$v_i = \begin{cases} 1 \text{ if vertex i is in the MIS} \\ 0 \text{ otherwise} \end{cases}$$

$$\max \sum v_i$$

$$\text{s.t. } v_i + v_j \leq 1 \qquad \forall (i, j) \in E$$

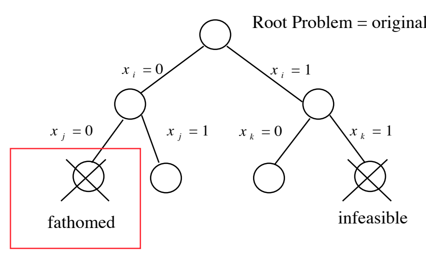
Example: Some Maximum Independent Set Cutting Planes



$$v_i + v_j + v_k \leq 1 \qquad \forall \text{ triangles i, j, k}$$

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

Value of a Good Feasible Solution Found Early



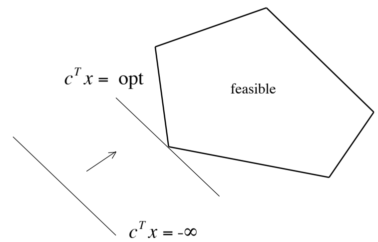
- Simulate seeding with optimal (computation is a proof of optimality)
- Note: proof of optimality depends on a good LP relaxation

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)

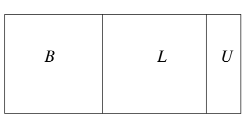


Linear Programming Algebra

What does a corner look like algebraically?

Ax=b

Partition *A* matrix into three parts



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 ($\ell_B \le x_B \le u_B$), this is a **basic feasible solution (BFS)**

Basic Feasible Solutions

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)

Set $x_B = B^{-1}b'$

In the common case $\ell = 0$, $u = \infty$ $(x \geq 0)$,

we have $b = b'$, $x_U = \emptyset$, $x_L = N$

x_B 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 $Ax + Ix_S = b$

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:

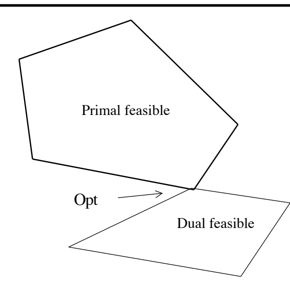
maximize $y^T b$

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



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.