15-853:Algorithms in the Real World

Linear and Integer Programming III
- Integer Programming
  - Applications
  - Algorithms

Integer (linear) Programming

minimize: $c^T x$
subject to: $Ax \leq b$
$x \geq 0$
$x \in Z^n$

Related Problems
- Mixed Integer Programming (MIP)
- Zero-one programming
- Integer quadratic programming
- Integer nonlinear programming

History

- Introduced in 1951 (Dantzig)
- TSP as special case in 1954 (Dantzig)
- First convergent algorithm in 1958 (Gomory)
- General branch-and-bound technique 1960 (Land and Doig)
- Frequently used to prove bounds on approximation algorithms (late 90s)

Current Status

- Has become "dominant" over linear programming in past decade
- Saves industry Billions of Dollars/year
- Can solve 10,000+ city TSP problems
- 1 million variable LP approximations
- Branch-and-bound, Cutting Plane, and Separation all used in practice
- General purpose packages do not tend to work as well as with linear programming --- knowledge of the domain is critical.

Subproblems/Applications

- **Facility location**
  Locating warehouses or franchises (e.g. a Burger King)
- **Set covering and partitioning**
  Scheduling airline crews
- **Multicomodity distribution**
  Distributing auto parts
- **Traveling salesman and extensions**
  Routing deliveries
- **Capital budgeting**
- **Other Applications**
  VLSI layout, clustering

Knapsack Problem

Integer (zero-one) Program:

maximize $c^T x$

subject to: $ax \leq b$

x **binary**

where:

b = maximum weight

c_i = utility of item i

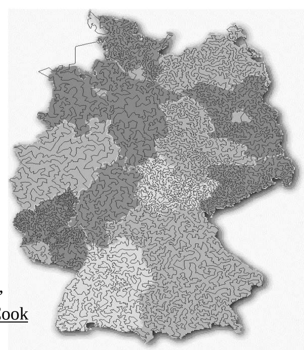
a_i = weight of item i

x_i = 1 if item i is selected, or 0 otherwise

The problem is NP-hard.

Traveling Salesman Problem

Find shortest tours that visit all of n cities.



courtesy: Applegate, Bixby, Chvátal, and Cook

Traveling Salesman Problem

minimize: $$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to:

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad 1 \leq i \leq n \quad \text{(out degrees = 1)}$$

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad 1 \leq j \leq n \quad \text{(in degrees = 1)}$$

$$t_i - t_j + n x_{ij} \leq n-1 \quad 2 \leq i, j \leq n \quad \text{(??)}$$

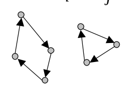
c_{ij} = distance from city i to city j

x_{ij} = 1 if tour visits i then j, and 0 otherwise (binary)

t_i = arbitrary real numbers we need to solve for

Traveling Salesman Problem

The last set of constraints: $t_i - t_j + nx_{ij} \le n-1 \quad \boxed{2} \le i, j \le n$
prevents "subtours":



Consider a cycle that goes from some node 4 to 5,
$t_4 - t_5 + nx_{4,5} \le n-1 \Rightarrow t_5 \ge t_4 + 1$
Similarly t has to increase by 1 along each edge of the cycle that does not include vertex 1.
Therefore, for a tour of length m that does not go through vertex 1, $t_4 \ge t_4 + m$, a contradiction.
Every cycle must go through vertex 1.
Together with other constraints, it forces one cycle.

Traveling Salesman Problem

Many "Real World" applications based on the TSP.

- They typically involve more involved constraints
- Not just routing type problems.
  Consider a drug company with k drugs they can make at a lab. They can only make the drugs one at a time. The cost of converting the equipment from making drug i to drug j is c_{ij}

Current best solutions are based on IP

- Applegate, Bixby, et. al., have solutions for more than 15K cities in Germany
  > 150,000 CPU hours (more info)
- Involves "branch-and-bound" and "cutting planes"

Set Covering Problem

Find cheapest sets that cover all elements

Courtesy: Darmstadt University of Technology

Set Covering and Partitioning

Given m sets and n items:

$$A_{ij} = \begin{cases} 1, & \text{if set j includes item i} \\ 0, & \text{otherwise} \end{cases}$$
$$c_i = \text{cost of set j}$$
$$x_j = \begin{cases} 1, & \text{if set j is included} \\ 0, & \text{otherwise} \end{cases}$$

Columns = sets
Rows = items

Set covering: **minimize:** $c^T x$ **subject to:** $Ax \ge 1$, x binary

Set partitioning: **minimize:** $c^T x$ **subject to:** $Ax = 1$, x binary

Set Covering and Partitioning

set	members	cost
s_1	{a,c,d}	.5
s_2	{b,c}	.2
s_3	{b,e}	.3
s_4	{a,d}	.1
s_5	{c,e}	.2
s_6	{b,c,e}	.6
s_7	{c,d}	.2

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Best cover: s_2, s_4, s_5 = .5

Best partition: s_4, s_6 = .7

Set Covering and Partitioning

Applications:

- **Facility location.**
  Each set is a facility (e.g. warehouse, fire station, emergency response center).
  Each item is an area that needs to be covered
- **Crew scheduling.**
  Each set is a route for a particular crew member (e.g. NYC->Pit->Atlanta->NYC).
  Each item is a flight that needs to be covered.

Constraints Expressible with IP

Many constraints are expressible with integer programming:

- logical constraints (e.g. x implies not y)
- k out of n
- piecewise linear functions

Constraints Expressible with IP

Logical constraints (x_1, x_2 binary):

Either x_1 or x_2 $\Rightarrow$ $x_1 + x_2 = 1$

If x_1 then x_2 $\Rightarrow$ $x_1 - x_2 \le 0$

Combining constraints:

Either $a_1x \le b_1$ or $a_2x \le b_2$ $\Rightarrow$ $\begin{array}{ll} a_1x - My & \le b_1 \\ a_2x - M(1-y) & \le b_2 \end{array}$

y is a binary variable, M needs to be "large",
a_1, a_2, and x can be vectors

Algorithms

1. Use a linear program
   - round to integer solution (what if not feasible?)
2. Search
   - Branch and bound (integer ranges)
   - Implicit (0-1 variables)
3. Cutting planes
   - Many variants

Important Properties

- LP solution is an upper bound on IP solution (assuming maximization)
- If LP is infeasible then IP is infeasible
- If LP solution is integral (all variables have integer values), then it is the IP solution.

Linear Programming Solution

1. Some LP problems will always have integer solutions
   - transportation problem
   - assignment problem
   - min-cost network flow

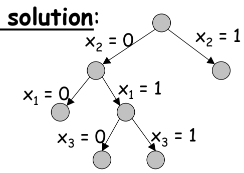
   These are problems with a unimodular matrix A.
   (unimodular matrices have det(A) = 1).
2. Solve as linear program and round. Can violate constraints, and be non-optimal. Works OK if
   - integer variables take on large values
   - accuracy of constraints is questionable

Branch and Bound

Lets first consider **0-1** programs.

Exponential solution: try all $\{0,1\}^n$

Branch-and-bound solution:



Traverse tree keeping current best solution.

If it can be shown that a subtree never improves on the current solution, **or** is infeasable, **prune** it.

Zero-One Branch and Bound

minimize: $z = c^T x$, subject to: $Ax \leq b, x \geq 0, x \in \{0,1\}^n$
Assume all elements of c are non-negative

function $\mathbf{ZO_r}(A, b, c, x_f, z^*)$
// x_f: a fixed setting for a subset of the variables
// z^* is the cost of current best solution
$x = x_f + 0$ // set unconstrained variables to zero
if ($cx \geq z^*$) **or** (no feasible completion of x_f) **return** z^*
if ($Ax \leq b$) **then return** cx
pick an unconstrained variable x_i from x
$z_0^* = \mathbf{ZO_r}(A, b, x_f \cup \{x_i = 0\}, c, z^*)$
$z_1^* = \mathbf{ZO_r}(A, b, x_f \cup \{x_i = 1\}, c, z_0^*)$
return z_1^*

function $\mathbf{ZO}(A, b, c) = \mathbf{ZO_r}(A, b, c, \emptyset, 0)$

Zero-One Branch and Bound

Checking for feasible completions: check each constraint and find if minimum of left is greater than right.

Example:

$x_f = \{x_1 = 1, x_3 = 0\}$

and one of the constraints is

$3x_1 + 2x_2 - x_3 + x_4 \leq 2$

then

$3 + 2x_2 - 0 + x_4 \leq 2$

$2x_2 + x_4 \leq -1$

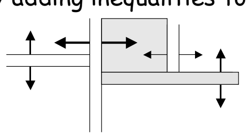
which is impossible.

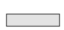
Integer Branch and Bound

The zero-one version is sometimes called "implicit enumeration" since it might enumerate all possibilities.

An integer version **cannot** branch on all possible integer values for a variable. Even if the integer range is bounded, it is not practical.

Will "bound" by adding inequalities to split the two branches.



Since solutions are integral, each split can remove a strip  of width 1

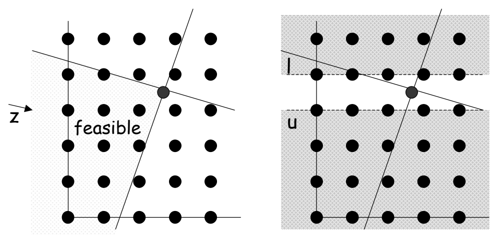
Integer Branch and Bound

maximize: $z = c^T x$, subject to: $Ax \leq b, x \geq 0, x \in Z^n$

function $\mathbf{IP_r}(A_e, b_e, c, z^*)$
// A_e, b_e are A and b with additional constraints
// z^* is the cost of current best solution
$z, x, f = LP(A,b,c)$ // f indicates whether feasible
if not(f) **or** ($z < z^*$) **return** z^*
if (integer(x)) **return** z
pick a non-integer variable x_i' from x
$z_l^* = \mathbf{IP}$(extend A_e, b_e with $x_i \leq \lfloor x_i' \rfloor$, c, z^*)
$z_g^* = \mathbf{IP}$(extend A_e, b_e with $-x_i \leq -\lceil x_i' \rceil$, c, z_l^*)
return z_g^*

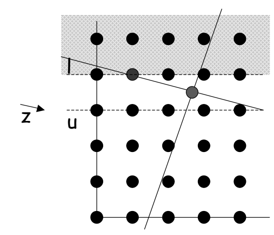
function $\mathbf{IP}(A, b, c) = \mathbf{IP_r}(A, b, c, -\infty)$

Example



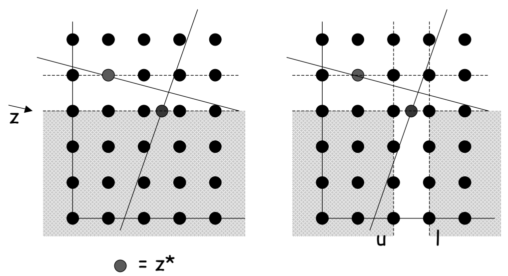
Find optimal solution.
Cut along y axis, and make two recursive calls

Example



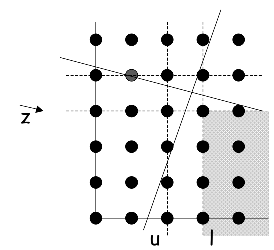
Find optimal solution.
Solution is integral, so return it as current best z*

Example



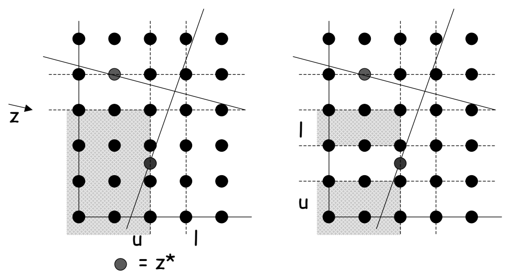
Find optimal solution. It is better than z*.
Cut along x axis, and make two recursive calls

Example



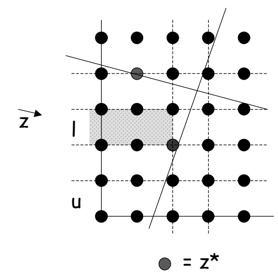
Infeasible, Return.

Example



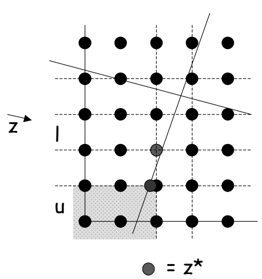
Find optimal solution. It is better than z*.
Cut along y axis, and make two recursive calls

Example



Find optimal solution. Solution is integral and better than z*. Return as new z*.

Example



Find optimal solution. Not as good as z*, return.

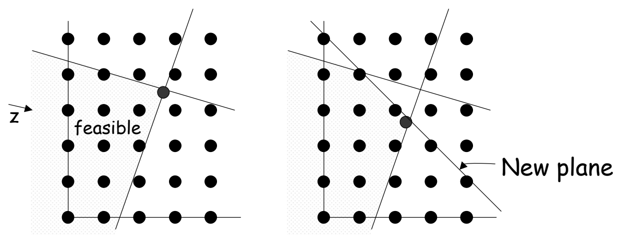
Cutting Plane

General Algorithm:

function IP(*C*, c)
    x = LP(*C*,c)
    while not(integer(x) or infeasible(x))
        add constraints to *C*
        x = LP(*C*,c)

The constraints must have the properties that
1. they do not cut off any feasible integer solutions
2. they do cut off the current optimal linear solution

Cutting Plane



Note that we are removing a corner, and no integer solutions are being excluded.

Picking the Plane

Method 1: Gomory cuts (1958)

- Cuts are generated from the LP Tableau
  Each row defines a potential cut
- Guaranteed to converge on solution
- General purpose, but inefficient in practice

Method 2: problem specific cuts

- Consider the problem at hand and generate cuts based on its structure
- Not general purpose, but can work very well in practice

Underconstraint Problems

It is possible to underconstrain the initial problem, and add constraints (cuts) later as needed.

For example, in TSP formulation, one could leave out the subtour elimination constraints. Constraints would only be added when a subtour is formed, and then only enough to break the subtour.
Such constraints can take advantage of properties of graphs and tours. (more info)

This technique is used frequently in practice.

Practical Developments

- **Good formulations**, heuristics and theory
  Goal: to get LP solution as close as possible to IP solution
  Disaggregation, adding constraints (cuts)
- **Preprocessing**
  Automatic methods for reformulation
  Some interesting graph theory is involved
- **Cut generation** (branch-and-cut)
  Add cuts during the branch-and-bound
- **Column generation**
  Improve formulation by introducing an exponential number of variables.