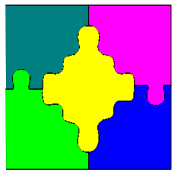
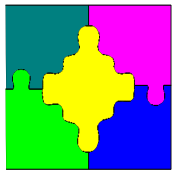


Integer Programming

Using linear programming to solve discrete problems



Solving Discrete Problems

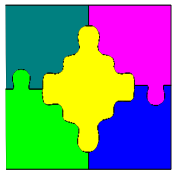
Linear programming solves *continuous* problem
—problems over the real numbers.

For the remainder of the course we will look at how to solve problems in which variables must take *discrete* values
—problems over the integers.

Discrete problems are harder than continuous problems.

We already saw: Finite Domain Propagation
We will look at

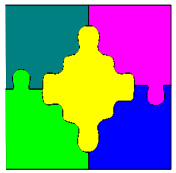
- Network Flow Problems
- Integer Programming
- Boolean Satisfiability
- Local Search



Mixed Integer Programming (MIP)

This week we will look at

- Different kinds of MIP programs
- Modelling with MIP
- Methods for solving MIP problems
  - Branch & Bound
  - Cutting Plane
  - Branch & Cut



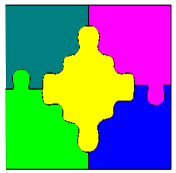
Simple Example, revisited

Reconsider our example for production plan optimization. This time we assume that B stands for “Book Cases”, C for “Chairs”, D for “Desks”.

	B	C	D	Capacity
Finishing	2.0	2.0	1.0	30.0
Labor	1.0	2.0	3.0	25.0
Machine Time	2.0	1.0	1.0	20.0
Profit	3.0	1.0	3.0	

```
maximize
profit      f(x*) = 3.0 x[b] + 1.0 x[c] + 3.0 x[d]
subject to  c(x*) = 2.0 x[b] + 2.0 x[c] + 1.0 x[d] <= 30.0
                and 1.0 x[b] + 2.0 x[c] + 3.0 x[d] <= 25.0
                and 2.0 x[b] + 1.0 x[c] + 1.0 x[d] <= 20.0
                and x[b]>=0 and x[c]>=0 and x[d]>=0
```

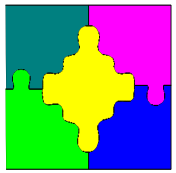
The optimum solution for this problem was to produce 7 bookcases and 6 desks.



Integer Problems

Now assume some coefficients would have been slightly different.
For example, each bookcase needs 1.5 units of labor.

	B	C	D	Capacity
Finishing	2.0	2.0	1.0	30.0
Labor	1.5	2.0	3.0	25.0
Machine Time	2.0	1.0	1.0	20.0
Profit	3.0	1.0	3.0	



Integer Problems

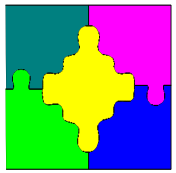
We apply Simplex and obtain the final tableau

z == 36.6667 - 0.6667 s[l] - 1.0 s[m] - 1.333 x[c]

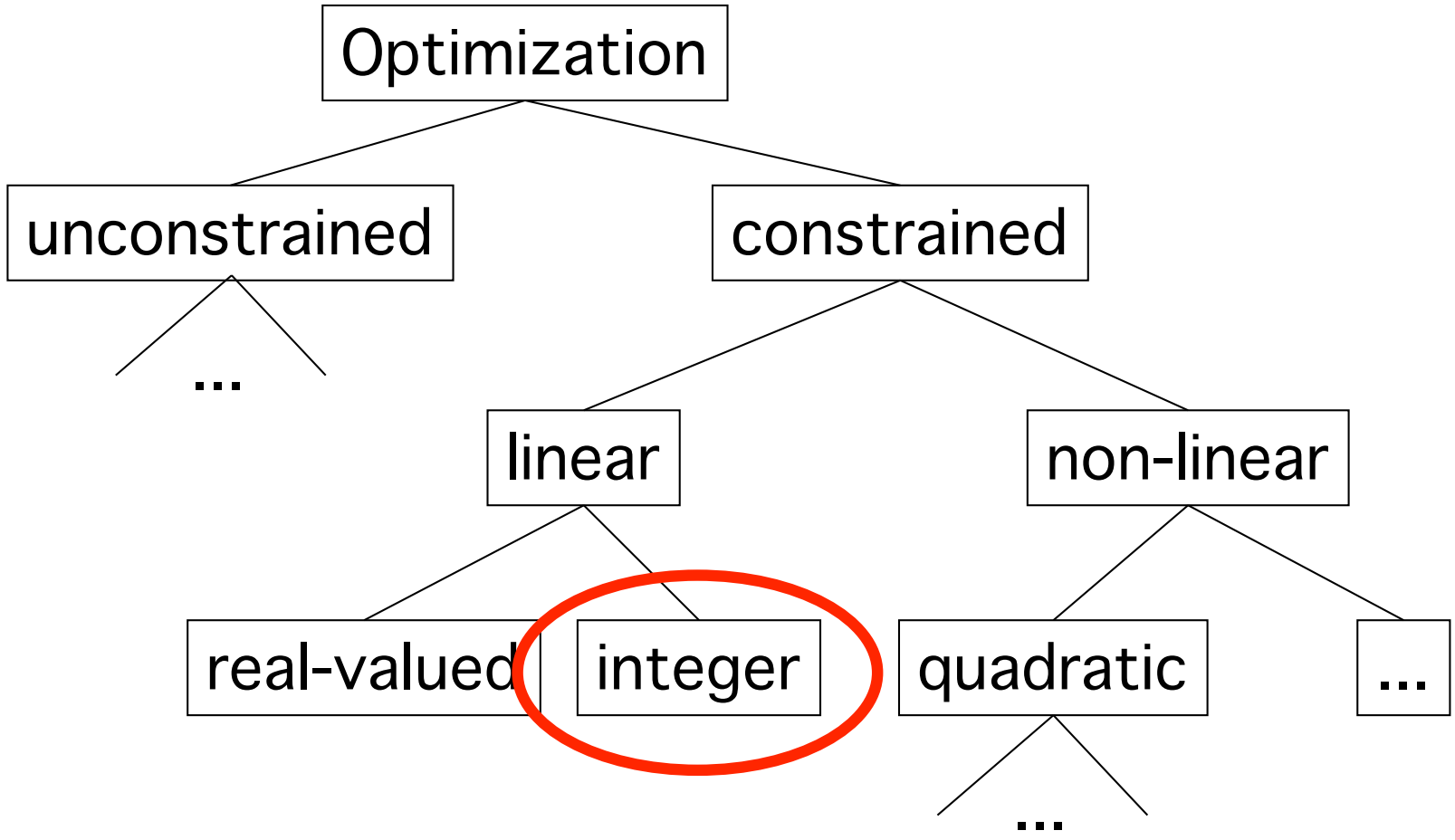
s[f] == 10 + s[m] - x[c]
x[d] == 4.4444 - 0.4444 s[l] + 0.3333 s[m] - 0.5556 x[c]
x[b] == 7.7778 +0.2222 s[l] - 0.6667 s[m] - 0.2222 x[c]

This makes little sense as a solution, since we can hardly manufacture 4.444 desks (maybe we can, but who would buy 0.444 of them).

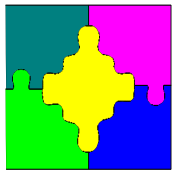
The problem that we are facing is that we need the problem variables to assume integer values for a feasible solution.
Such problems are called integer programming problems.



Classes of Optimization Problems



$$\max_{\vec{x}} \quad f(\vec{x}) \quad \text{subject to} \quad C(\vec{x})$$



Integer Programming Problems

$$\max_{\vec{x}} \quad f(\vec{x}) \quad \text{subject to} \quad C(\vec{x})$$

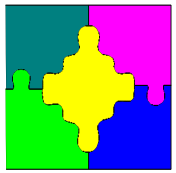
where $f(\vec{x}) = f(x_1,\ldots x_n)$

is a linear function $f : R^n \rightarrow R$ and

$C(\vec{x}) = c_1(x_1,\ldots x_n) \wedge \ldots \wedge c_k(x_1,\ldots,x_n)$

is a conjunction of linear inequalities

and the x_i are required to be integer.



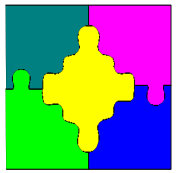
...the bad news

Integer linear problems are significantly harder to solve than linear problems on reals.

No general algorithm is known that allows to optimize a solution by directly moving from a feasible solution to an improved feasible solution.

In fact, integer linear programming is known to be NP-complete.

... It seems easy to formulate a TSP as an IP...



TSP as an Integer Program

$$\text{minimize} \quad \sum\sum c_{ij} x_{ij}$$

$$\text{subject to} \quad \forall i: \; 1 = \sum_{j=1\ldots n} x_{ij} \quad \wedge \quad \forall j: \; 1 = \sum_{i=1\ldots n} x_{ij}$$

$$\text{where} \quad x_{ij} = \begin{cases} 1 & \text{if tour leads from city i to city j} \\ 0 & \text{otherwise} \end{cases}$$

The constraints ensure each city is reached and left

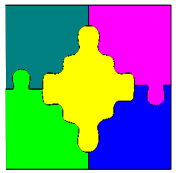
Is this correct??

We also need to ensure that there are no sub-tours, e.g. 1-5-2-1 3-4-3

$$u_i - u_j + n x_{ij} \leq n - 1 \quad for\ i = 2,\ldots,n, j = 2,\ldots,n\ and\ i \neq j$$

$$all\ u_i \geq 0$$

Any sub-tour that does not contain city 1 will fail this constraint



Mixed Integer Programming (MIP) Problems

$$\max_{\vec{x}} \quad f(\vec{x}) \quad \text{subject to} \quad C(\vec{x})$$

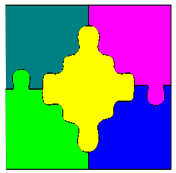
where $f(\vec{x}) = f(x_1, \ldots x_n)$

is a linear function $f : R^n \rightarrow R$ and

$C(\vec{x}) = c_1(x_1, \ldots x_n) \wedge \ldots \wedge c_k(x_1, \ldots, x_n)$

is a conjunction of linear inequalities

and some x_i are required to be integer - valued.



0-1 Integer Problems

$$\max_{\vec{x}} \quad f(\vec{x}) \quad \text{subject to} \quad C(\vec{x})$$

where $f(\vec{x}) = f(x_1,\ldots x_n)$

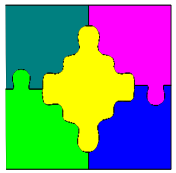
is a linear function $f : R^n \rightarrow R$ and

$C(\vec{x}) = c_1(x_1,\ldots x_n) \wedge \ldots \wedge c_k(x_1,\ldots,x_n)$

is a conjunction of linear inequalities

and $\forall i : x_i \in \{0,1\}$

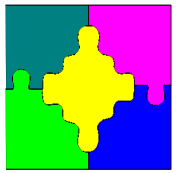
These are also called Binary Integer Programming (BIP) problems



Modelling in MIP

Operation researchers have developed a large number of clever techniques for modelling problems that look non-linear as MIP problems.

Many of the tricks rely on the use of 0-1 variables to model a binary choice.



Modelling Or Constraints

To model two constraints of which at least one must be satisfied (*f* or *g*)

$$f(\vec{x}) \leq 0$$

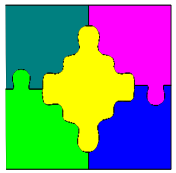
$$g(\vec{x}) \leq 0$$

Use a 0-1 variable *y* and reformulate as

$$f(\vec{x}) \leq M \cdot y$$

$$g(\vec{x}) \leq M \cdot (1 - y)$$

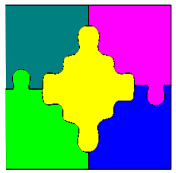
where *M* is a sufficiently large number



Modelling K out of N Constraints Hold

Exercise: How can we model that K out of the following N constraints hold

$$f_1(\vec{x}) \leq 0, \quad \cdots \quad , f_N(\vec{x}) \leq 0$$



If Then Constraints

To model an if-then relation between two constraints (*A* =>*B)*

$$f(\vec{x}) > 0$$

$$g(\vec{x}) \geq 0$$

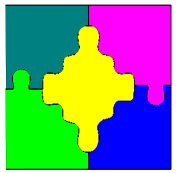
Recall that A => B is equivalent to (not A) or B.

Use a 0-1 variable *y* (*y*=0 if *f* is true) and reformulate as

$$-g(\vec{x}) \leq M \cdot y$$

$$f(\vec{x}) \leq M \cdot (1 - y)$$

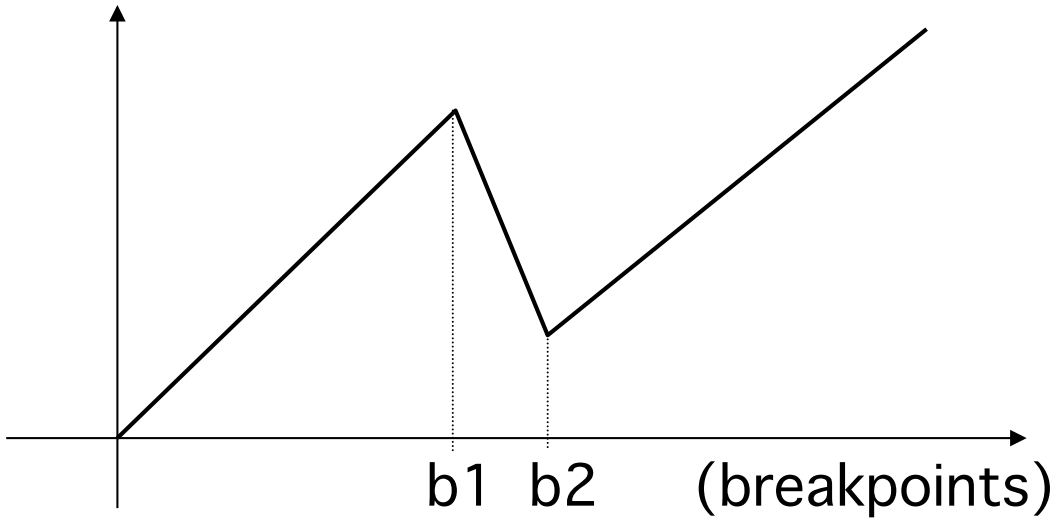
Where *M* is a sufficiently large number



Piecewise Linear Functions

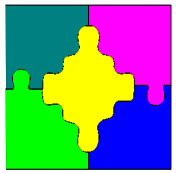
To model a piecewise linear function

- break it into linear segments,
- model the separate segments,
- "chain" the segments with an additional constraint



$$\text{for } b_1 \le x \le b_2 \qquad \text{with } 0 \le q \le 1$$

$$f(x) = q \cdot f(b_1) + (1-q) \cdot f(b_2)$$



Modelling Piecewise Linear Functions

(1) replace *f(x)* throughout the problem by *f* and add a constraint

$$f = z_1 \cdot f(b_1) + \ldots + z_n \cdot f(b_n)$$

(2) add a constraint for *x*

$$x = z_1 \cdot b_1 + \ldots + z_n \cdot b_n$$

(3) add the following constraints

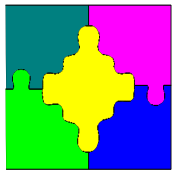
$$z_1 \le y_1, \quad z_2 \le y_1 + y_2, \quad z_3 \le y_2 + y_3, \quad \ldots, \quad z_{n-1} \le y_{n-2} + y_{n-1}, \quad z_n \le y_{n-1}$$

$$\sum y_i = 1 \qquad \sum z_i = 1$$

$$\forall\, y_i \in \{0,1\}$$

$$\forall\, z_i \ge 0$$

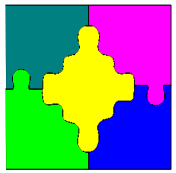
These constraints ensure that only two neighbouring *z*'s can have $z_i > 0$ and must add up to 1 => this models the piecewise interpolation



Modelling IP as 0-1 Integer Problem

Any IP problem can be modelled as a 0-1 Integer Problem

Exercise: How?

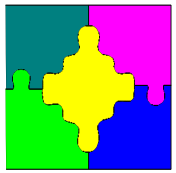


MIP Modelling in MiniZinc

- You can run a MIP solver on MiniZinc models using

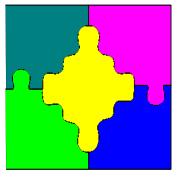
  ```
  -mzn -b mip model.mzn
  ```

- This assumes that the constraints are all linear.
- In order to effectively use a MIP solver we must learn how to model things with linear constraints.



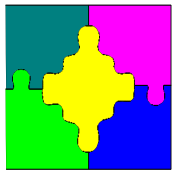
Boolean Modelling in MiniZinc

- Boolean constraints are modelled using 0..1 integers and linear constraints:
  - b1 = (b2 ∨ b3)
  - `b1 >= b2 /\ b1 >= b3 /\ b2 + b3 >= b1`
  - b1 = (b2 ∧ b3)
  - `b2 >= b1 /\ b3 >= b1 /\ b1+1 >= b2+b3`
  - b1 = (b2 -> b3)
  - `b1 >= 1 - b2 /\ b1 >= b3 /\`
    `1-b2 + b3 >= b1`
- Question: how do you model b2 ∨ b3?
  -



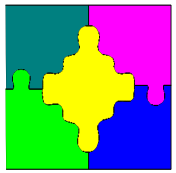
Reified Linears in MIP

- Recall that x1 <= x2 ∨ x1 <= x3 becomes
  - b1 ⇔ x1 <= x2 ∧ b2 ⇔ x1 <= x3 ∧ (b1 ∨ b2)
- b ⇔ sum(i in 1..n)(a[i]*x[i]) <= c
  - Let l = sum(i in 1..n)(lb(a[i]*x[i]))
  - Let u = sum(i in 1..n)(ub(a[i]*x[i]))
- When b is true (1) force the ineq to hold
  - `sum(i in 1..n)(a[i]*x[i]) <= u + b*(c - u)`
- When b is false (0) force reverse to hold
  - `sum(i in 1..n)(a[i]*x[i]) >= c+1+ b*(l-c-1)`



Modelling Disequality

- x != y ????
  - Model x,y variables as Booleans (x = d)
  - ```
    var 1..n:x
    array[1..n] of var 0..1:xd;
    constraint sum(i in 1..n)(i*xd[i]) = x;
    constraint sum(i in 1..n)(xd[i]) = 1;
    ```
  - Now we can model disequality
  - ```
    constraint forall(i in 1..n)(
                      xd[i] + yd[i] <= 1);
    ```
- With these 0..1 variables we can model many things

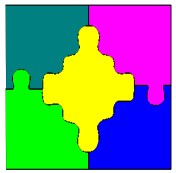


Modelling Element

- element(i, [a_1, …, a_n], x) $\Leftrightarrow$ x = a_i
  - i = j -> x = a_j
- `array[1..n] of var 0..1:id;`

  `constraint`

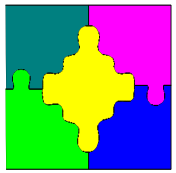
  `x = sum(j in 1..n)(id[j] * a`$_j$`);`



Exercise: Modelling alldifferent

- How can you model
  - `array[1..n] of var 1..m: x;`
    `constraint alldifferent(x);`

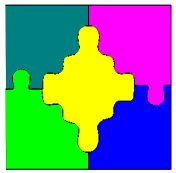
  using only linear constraints?



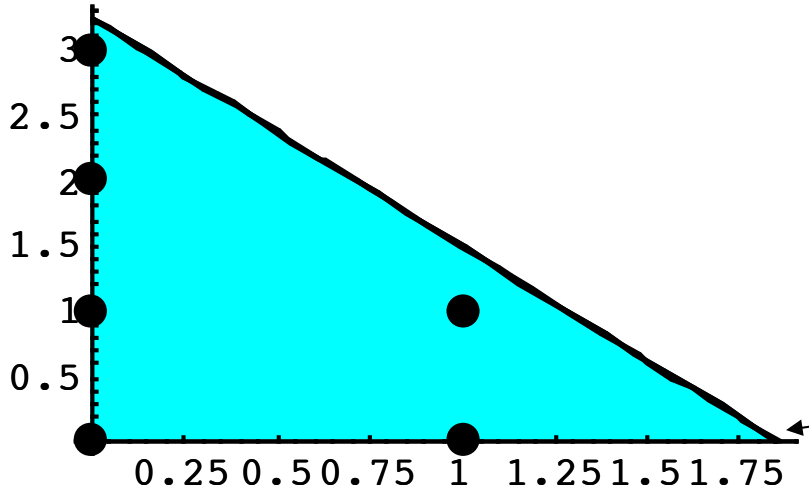
Mixed Integer Programming (MIP)

This week we will look at

- Different kinds of MIP programs
- Modelling with MIP
- Methods for solving MIP problems



Linear Relaxation

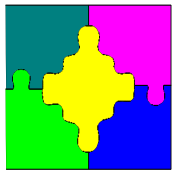


max 21x+11y
s.t. 7x+4y<=13

optimum

The first idea might be to try rounding of the optimal fractional solution. The corresponding problem in which the variables are not required to take integer values is called the linear relaxation.

As the above example proves, this cannot be guaranteed to work:
(2/0) is infeasible
(1/0) is feasible, but not optimal



Solving MIP Problems

Problem: Linear Relaxation does not work

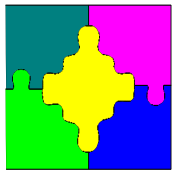
Observation: (1) All feasible points for MIP are also feasible for the relaxation
(2) If the optimum of the relaxation is integer-valued,
it also is the solution of the original MIP.

Idea: Enumerate the feasible points of the relaxed problem

Challenge: This must be done cleverly, since the number of feasible points may be huge.

Methods: based on repeatedly solving improved relaxations (with simplex)

- Branch & Bound: splits the search space into sub-problems
- Cutting Plane: removes non-integer optimums
- Branch & Cut: split & remove non-integer optimums



Branch & Bound

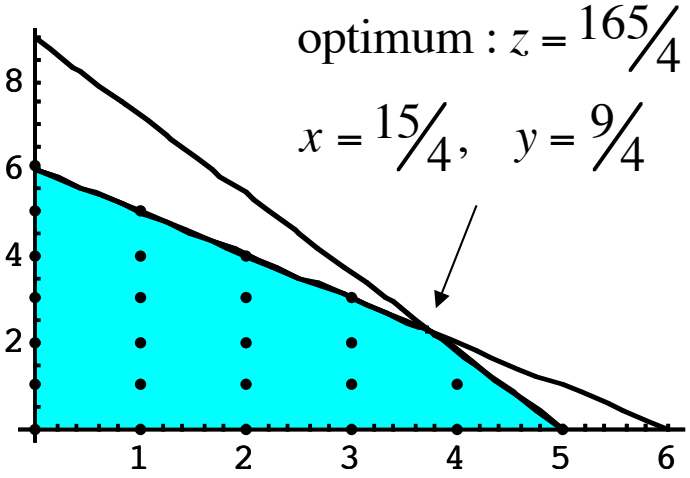
Let *P* be a linear integer program.
To solve *P*

1. Solve *P*s linear relaxation *P'*.
2a. If the solution of *P'* is integer-valued, it is the optimum of *P*
2b. If the solution contains a fractional assignment $x_i = v$ to an integer variable create two sub-problems:

$$P'_{below}: \quad P \wedge x_i \leq \lfloor v \rfloor$$

$$P'_{above}: \quad P \wedge x_i \geq \lceil v \rceil$$

3. Solve the sub-problems recursively.
4. The solution to *P* is the better solution of *P'*$_{below}$ *and* *P'*$_{above}$



$$\max z = 8x + 5y$$

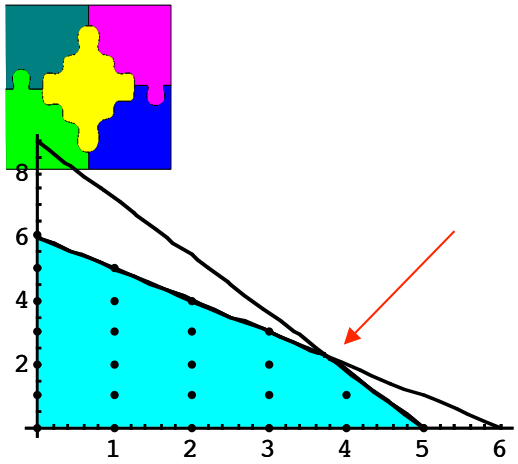
such that

$$x + y \leq 6$$

$$9x + 5y \leq 45$$

$$x, y \geq 0, \text{integer}$$

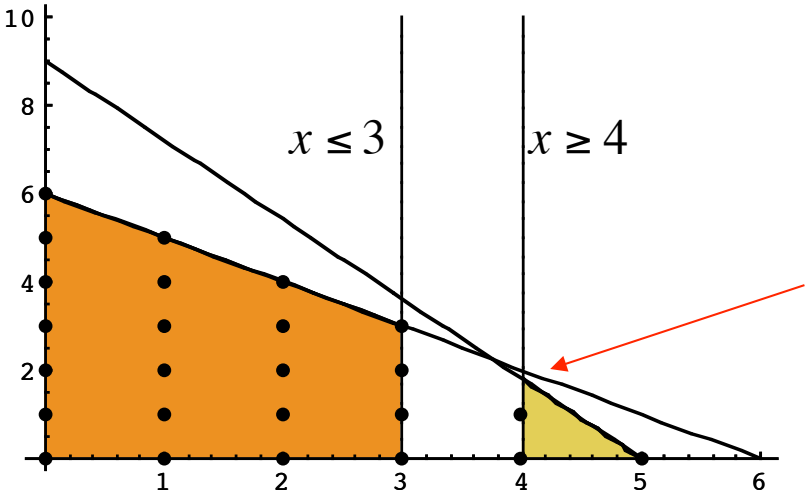
Branch & Bound



$$\max z = 8x + 5y$$
$$\text{such that} \quad x + y \le 6$$
$$9x + 5y \le 45$$
$$x, y \ge 0$$

optimum : $z = 165/4$

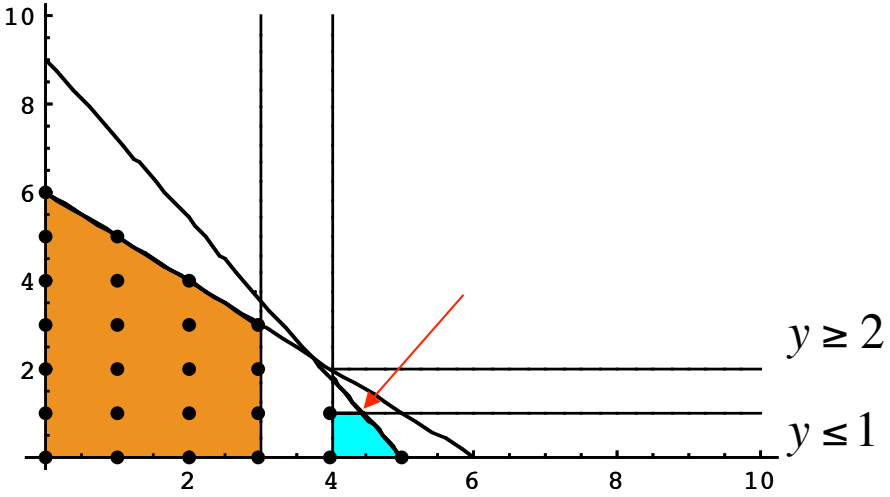
$x = 15/4, \quad y = 9/4$



$$\max z = 8x + 5y$$
$$\text{such that} \quad x + y \le 6$$
$$9x + 5y \le 45$$
$$x \ge 4 \ (*)$$
$$x, y \ge 0$$

optimum : $z = 4$

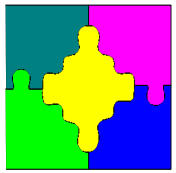
$x = 4, \quad y = 9/5$



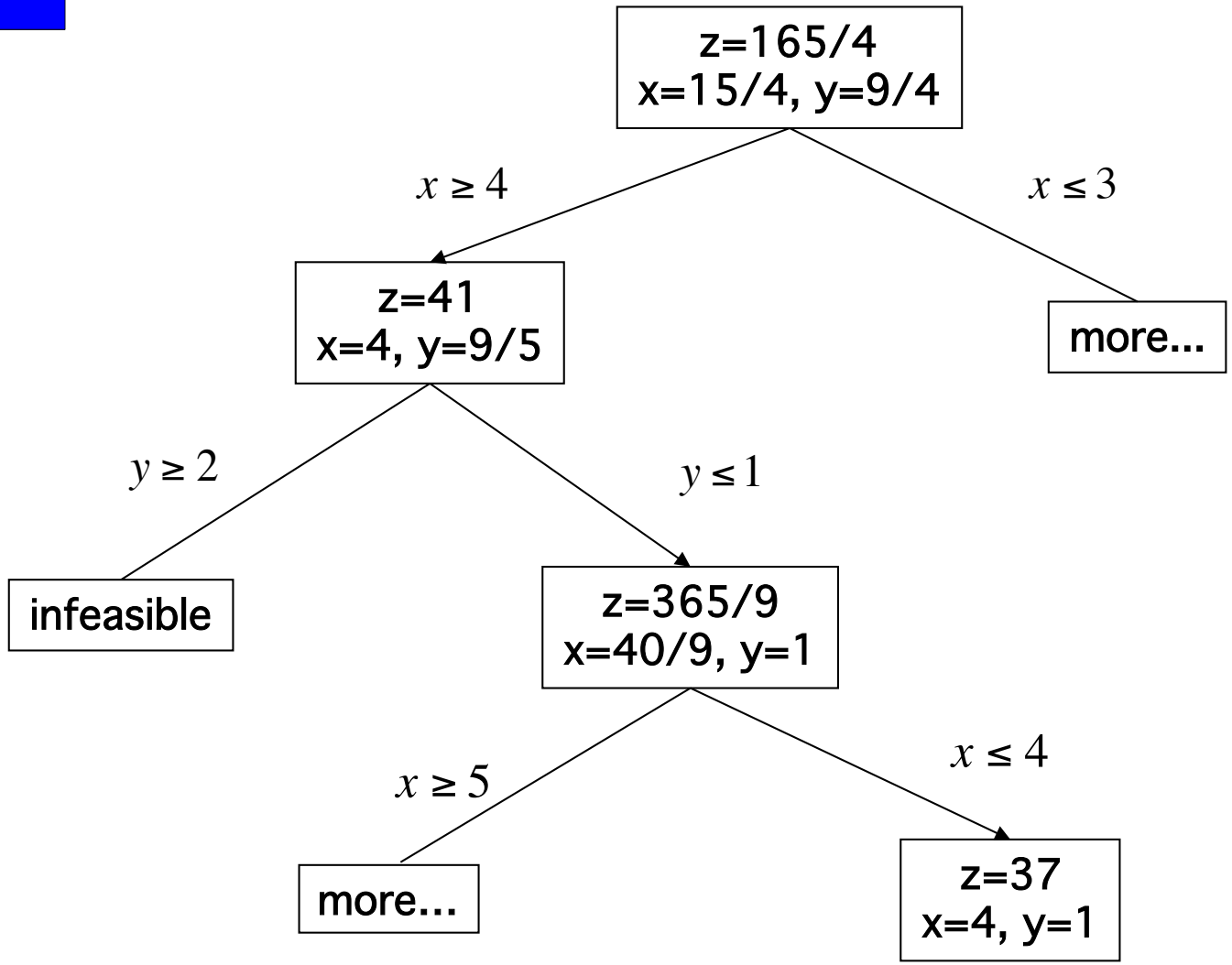
$$\max z = 8x + 5y$$
$$\text{such that} \quad x + y \le 6$$
$$9x + 5y \le 45$$
$$x \ge 4$$
$$y \le 1 \ (*)$$
$$x, y \ge 0$$

optimum : $z = 365/9$

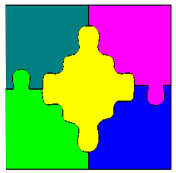
$x = 40/9, \quad y = 1$



Branch & Bound Decision Tree



Solutions of a sub-problem in which all integers are assigned an integer are called Candidate Solutions



Fathomed Sub-Problems

Not every sub-problem (branch) needs to be explored.

In three cases, a sub-problem does not need to be explored further.

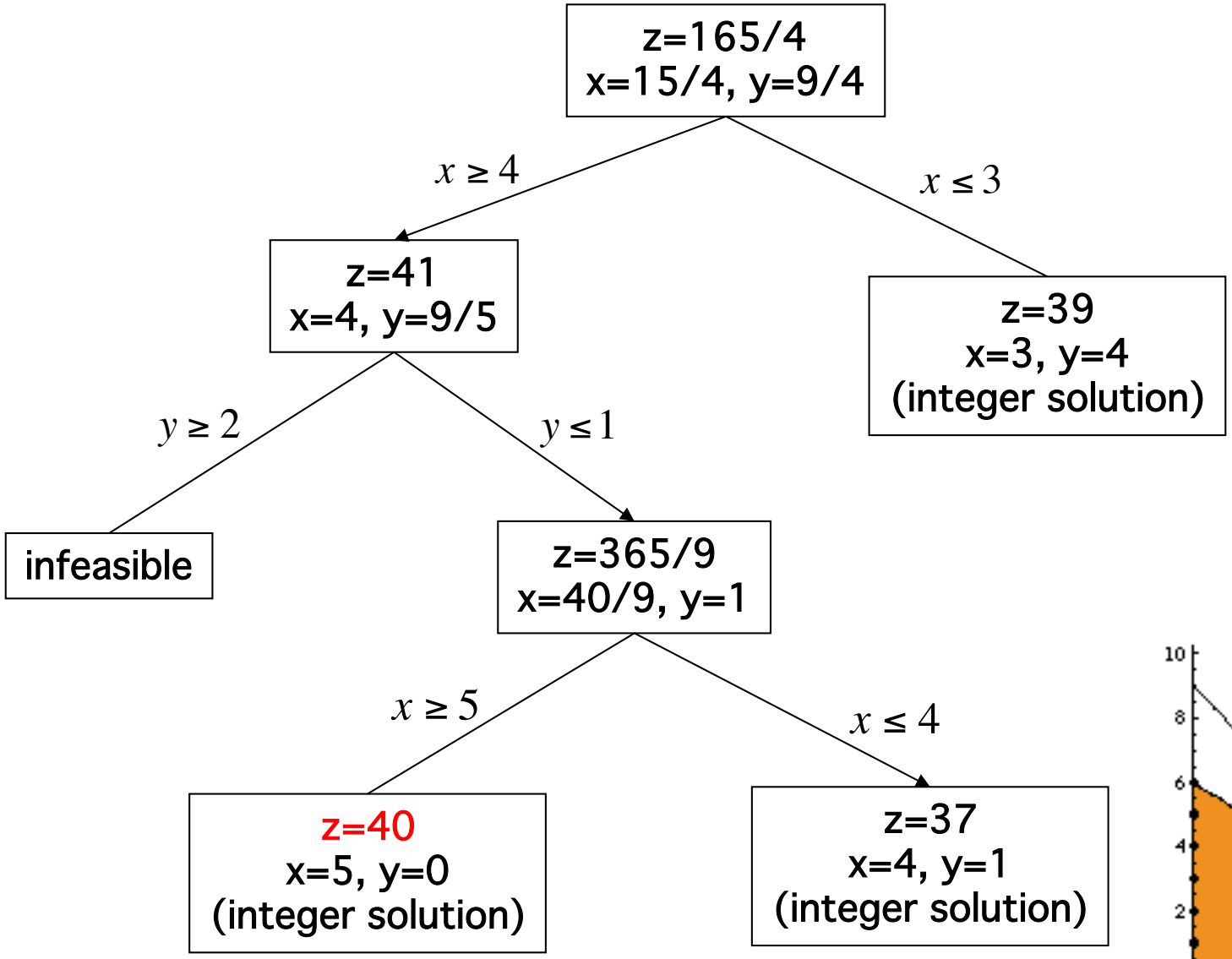
- it is infeasible
- its optimum solution is integer-valued
- the optimum of its relaxation is smaller than that of some previously found candidate.

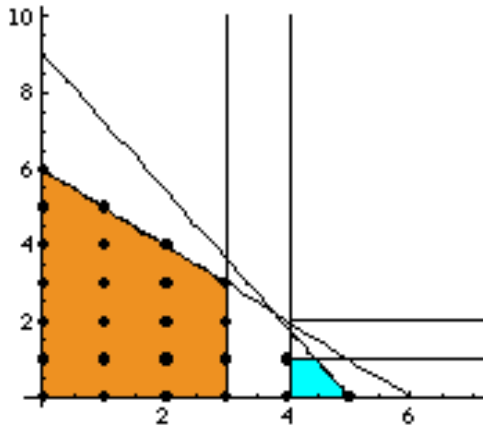
  This is because

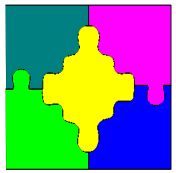
  - the candidate solution provides a lower bound for the original problem
  - the optimum of the relaxation provides an upper bound for the corresponding integer problem.

Such sub-problems are called fathomed.

Final Decision Tree







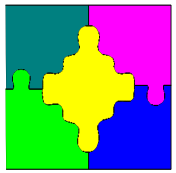
Backtracking vs. Jumptracking

The order in which sub-problems are expanded is not fixed.

Two strategies are common

- Backtracking (aka LIFO / last-in-first-out)
  - Expand the most recently created sub-problem first
  - This results in depth-first exploration.

- Jumptracking
  - obtain upper bounds for all sub-problems through relaxation
  - then expand the sub-problem with the best upper bound

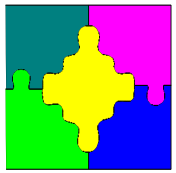


Choosing the Branching Variable

The other choice is which variable to branch upon.

A standard rule is to branch on the variable that has the greatest economic importance. That is has the largest absolute coefficient in the objective function.

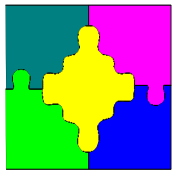
The choice of branching variable and of sub-problem expansion order can make a huge difference to efficiency.



Branch & Bound

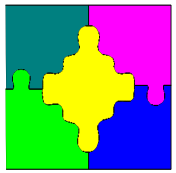
Branch and bound is a very general technique for solving discrete problems.

- MIPs can be solved by branch & bound--we only need to branch on the integer variables.
- It is obviously easier to solve 0-1 IPs, because branching on x only needs to consider two cases: x=0 and x=1.
- It can be used whenever it is easy to solve a relaxation of the problem.



Homework

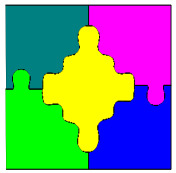
- Read Chapter 9 of Winston.
- Verify that the encoding of TSP is correct.
- Write a MIP model for the Set Covering Problem.
- Write a MIP model for the Warehouse Allocation Model.



Mixed Integer Programming (MIP)

This week we will look at

- Different kinds of MIP programs
- Modelling with MIP
- Methods for solving MIP problems
  - Branch & Bound
  - Cutting Plane
  - Branch & Cut



Cutting Planes

Idea: If the solution of the LP relaxation is not integer-valued we can find an additional constraint (called "cut") that

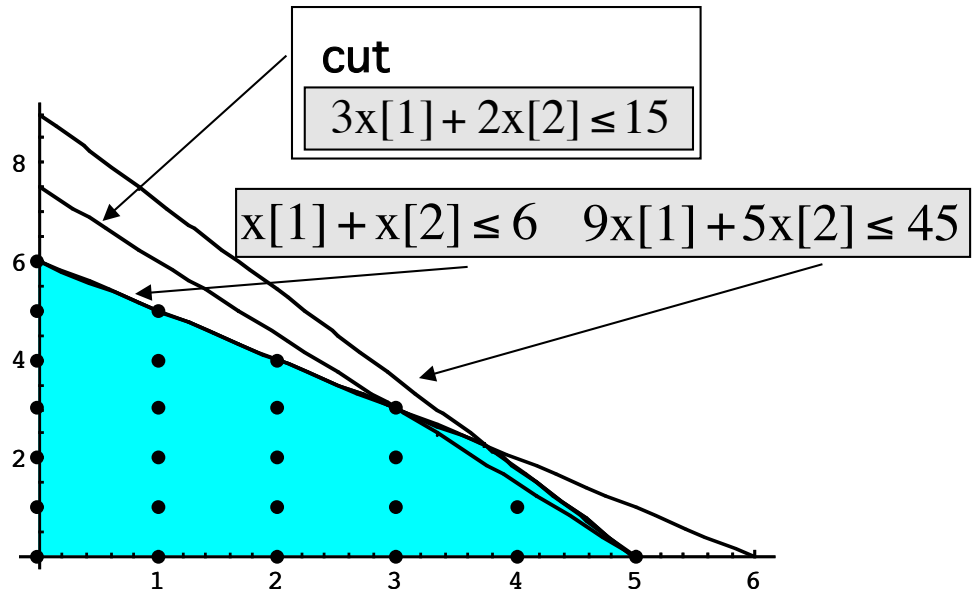
- removes the current optimum from the feasible region
- does not remove any feasible point for the original IP

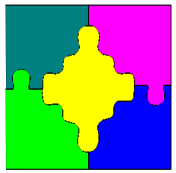
Problem:

maximize $8x[1]+5x[2]$

subject to $9x[1]+5x[2] \le 45$

and $x[1]+x[2] \le 6$





Finding the Cut (1)

Problem: maximize $8x[1] + 5x[2]$

$$x[1] + x[2] \le 6, \quad 9x[1] + 5x[2] \le 45$$

Initial Tableau:

$$z == 8x[1] + 5x[2]$$

$$s[1] == 6 - x[1] - x[2]$$

$$s[2] == 45 - 9x[1] - 5x[2]$$

Solution:

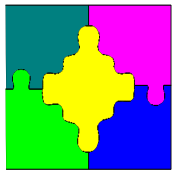
$$z == \frac{165}{4} - \frac{5\ s[1]}{4} - \frac{3\ s[2]}{4}$$

$$x[2] == \frac{9}{4} - \frac{9\ s[1]}{4} + \frac{s[2]}{4}$$

$$x[1] == \frac{15}{4} + \frac{5\ s[1]}{4} - \frac{s[2]}{4}$$

- Find an equation for a fractional basic variable

$$x[1] - \frac{5}{4}\ s[1] + \frac{1}{4}\ s[2] == \frac{15}{4}$$



Finding the Cut (2)

(1) rearrange fractional constraint

$$x[1] - \frac{5}{4}\,s[1] + \frac{1}{4}\,s[2] == \frac{15}{4}$$

(2) rewrite fractional coefficients into fractional part and integer part using |_ x _|

$$x[1] - 2s[1] + \frac{3}{4}s[1] + \frac{1}{4}s[2] == 3 + \frac{3}{4}$$

(3) move fractional parts to right hand side

$$x[1] - 2s[1] - 3 == \frac{3}{4} - \frac{3}{4}s[1] - \frac{1}{4}s[2]$$

(4) Generate cut as (RHS <= 0)

$$\frac{3}{4} - \frac{3}{4}s[1] - \frac{1}{4}s[2] \le 0$$

(5) Add slack to cut

$$s[3] == -\frac{3}{4} + \frac{3}{4}s[1] + \frac{1}{4}s[2]$$

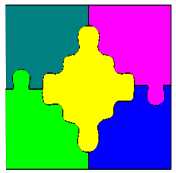
(6) Add cut to tableau & resolve

$$z == \frac{165}{4} - \frac{5\,s[1]}{4} - \frac{3\,s[2]}{4}$$
$$x[2] == \frac{9}{4} - \frac{9\,s[1]}{4} + \frac{s[2]}{4}$$
$$x[1] == \frac{15}{4} + \frac{5\,s[1]}{4} - \frac{s[2]}{4}$$
$$s[3] == -\frac{3}{4} + \frac{3}{4}s[1] + \frac{1}{4}s[2]$$

Note: The cut equation

$$3x[1] + 2x[2] == 15$$

is obtained by substituting the definitions of the slack variables in s[3]==0.



Finding the Cut (3)

The cut generated by this method has two properties

(1)The current optimal solution to the LP relaxation does not satisfy the cut $\frac{3}{4} - \frac{3}{4}s[1] - \frac{1}{4}s[2] \le 0$

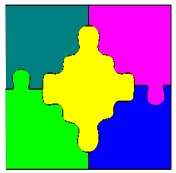
The current solution has s[1]=s[2]=0

(2) Any feasible solution for the original MIP satisfies the cut

Consider a feasible solution. This must satisfy

$$x[1] - 2s[1] - 3 == \frac{3}{4} - \frac{3}{4}s[1] - \frac{1}{4}s[2]$$

In any feasible solution s[1] s[2] ≥ 0, so the RHS < 1.
Since x[1] and s[1] must be integers the LHS is an integer.
Thus the LHS=RHS ≤ 0.



Cutting Plane Algorithm

(1)	Solve the LP relaxation
(2a)	If the solution to the relaxation is integer-valued optimum is found
(2b)	Otherwise add new cut to tableau of LP relaxation
(3)	goto step 1

Note: the row of the cut constraint (after adding slack) in the tableau has a negative intercept.
Simplex as we know it is unsuitable for this problem.

$$z == \frac{165}{4} - \frac{5\,s[1]}{4} - \frac{3\,s[2]}{4}$$

$$x[2] == \frac{9}{4} - \frac{9\,s[1]}{4} + \frac{s[2]}{4}$$

$$x[1] == \frac{15}{4} + \frac{5\,s[1]}{4} - \frac{s[2]}{4}$$

$$s[3] == -\frac{3}{4} + \frac{3}{4}s[1] + \frac{1}{4}s[2]$$

Dual Simplex

(1) If all intercepts are positive terminate.

(2) Choose the basic variable with the most negative intercept to leave the basis. The corresponding row is the pivot row.

(3) Compute the following ratio for each variable x[j] with positive coefficient in the pivot row:

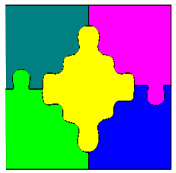
coefficient(x[j]) in objective / coefficient(x[j]) in pivot row

(4) Choose the variable with the smallest absolute ratio to enter the basis.

(5) Pivot the entering variable into the basis

(6a) If there is a row with a negative intercept and only negative coefficients terminate with "infeasible"

(6b) Otherwise go to step (1).

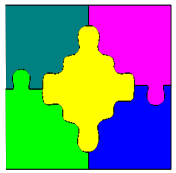


Dual Simplex

(1) Dual Simplex takes an optimal infeasible solution and maintains optimality (non-neg. coefficients in the cost row) while moving towards a feasible solution (non-neg. coefficients in the RHS) .

(2) It is really the Simplex algorithm applied to the dual problem.

(3) It is used in both branch-and-cut and cutting plane algorithms when a constraint is added to an optimal tableau and this causes infeasibility.



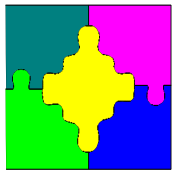
Implicit Enumeration for 0-1 IP

Generate a search tree for the problem variables
At each node some variables have fixed values,
others are “free” (value not yet known).
Prune the search tree with the following checks:

(1) *Check Bounds*
Assign values to the free variables that maximize/minimize the objective function. This is an upper bound.

(1a) If this valuation is feasible, it must be the optimum
(for the sub-tree with this root)
Keep as a candidate solution.

(1b) If the objective value for this valuation is
less than some candidate,
do not consider this node further.



Branch-and-Cut

The cutting plane approach was one of the first techniques developed for solving IP problems (in 1958). However, branch and bound techniques developed in the 1960s and 1970s proved to better for most problems.

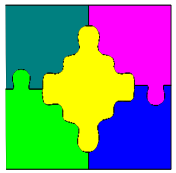
A combination of the two approaches branch-and-cut was developed in the 1980s for pure 0-1problems where it could solve much larger problems than branch and bound by itself. Now generalized to MIP.

Branch-and-cut works by doing a branch-and-bound search. But for the initial problem and each sub-problem it uses

- Automatic problem preprocessing
- Generation of cutting planes

to reduce the search space.

We shall only consider 0-1 problems.



Automatic Problem Preprocessing

Automatic preprocessing simplifies the constraints in the original problem and then sub-problems in order to make the problem easier to solve.

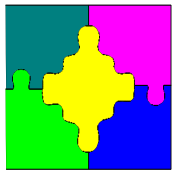
Technique 1: Detecting Infeasibility

If some constraint is unsatisfied when all the variables take their best value, then the problem is infeasible.

$$x_1+x_2-2x_3 \geq 3 \text{ because } 1(1)+1(1)-2(0) < 3.$$

The best value of a positive variable in the lhs of a $\leq$ is 0 and of a negative variable is 1. The reverse for a $\geq$.

As variables become fixed because of preprocessing and branching this can be used to detect infeasibility very early.



Automatic Problem Preprocessing

Technique 2: Fixing variables

If one value of a variable cannot satisfy the constraint when the other variables take their best value, then the variable should be fixed to the other value.

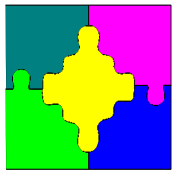
$3x_1 \leq 2$ implies $x_1=0$ since $3(1) > 2$

$5x_1-2x_2+x_3 \leq 2$ implies $x_1=0$ since $5(1)-2(1)+1(0) > 2$

These can cascade—do the following example

$3x_1+x_2-2x_3 \geq 2 \quad x_1+x_4+x_5 \leq 1 \quad -x_5+x_6 \leq 0$

This and other more complex tricks substantially reduce the number of variables, often by half.



Automatic Problem Preprocessing

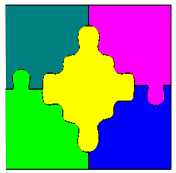
Technique 3: Eliminating Redundant Constraints

If some constraint is satisfied when all the variables take their worst value, then the constraint is redundant and can be removed without changing the solutions.

$x_1+x_2-2x_3 \leq 3$ because $1(1)+1(1)-2(0) \leq 3$.

The worst value of a positive variable in the lhs of a $\leq$ is 1 and of a negative variable is 0. The reverse for a $\geq$.

As variables become fixed because of preprocessing and branching significant numbers of constraints become redundant.



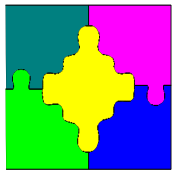
Automatic Problem Preprocessing

Technique 4: Tightening Constraints (Coefficient reduction)
Reduce the coefficients of a constraint if this reduces the feasible solutions in the relaxation but without removing 0-1 solutions.

$2x_1+3x_2 \leq 4$ can be tightened to $x_1+x_2 \leq 1$.

Many different techniques for doing this. See Chap 11Hillier & Leiberman.

This is closely related to generation of cutting planes. The difference is that tightening changes one of the constraints in the problem and does not add a new constraint.



Cutting Plane Generation

This adds a new constraint called the cutting plane which reduces the feasible solutions in the relaxation but does not remove 0-1 solutions.

One technique for generating a cutting plane from a ≤ constraint with only non-negative coefficients on the lhs and constants on the rhs is to find a subset of variables that form a minimum cover:

- The constraint is violated if every variable in the group is 1 and all other variables are set to 0 and
- The constraint becomes satisfied if any one of the variables in the group is changed to 0.

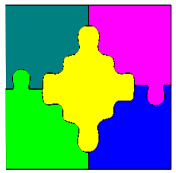
If there are N variables in the minimum cover then a cutting plane is

Sum of variables in group ≤ N-1.

$$6x_1+3x_2+5x_3+2x_4 \leq 10 \text{ implies } x_1+x_2+x_4 \leq 2.$$

There are many techniques for generating cutting planes.

In branch-and-cut many cutting planes are generated and added to the problem before branching.



Summary

This week we have looked at

- Different kinds of MIP programs
- Modelling with MIP
- Methods for solving MIP problems
  - Branch & Bound
  - Cutting Plane
  - Branch & Cut
- Modelling for MIP in MiniZinc