

CPS 570: Artificial Intelligence

**More search:
When the path to the solution
doesn't matter**

Instructor: Vincent Conitzer

Search where the path doesn't matter

- So far, looked at problems where the path was the solution
 - Traveling on a graph
 - Eights puzzle
- However, in many problems, we just want to find a goal state
 - Doesn't matter how we get there

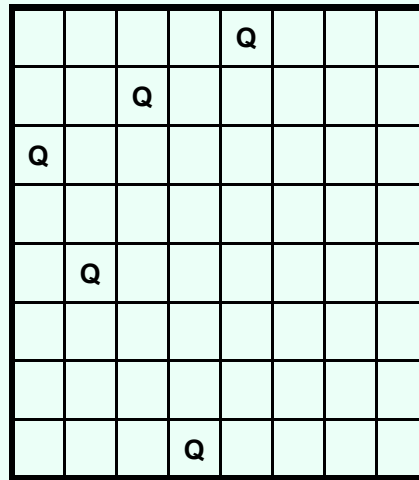
Queens puzzle

- Place eight queens on a chessboard so that no two attack each other

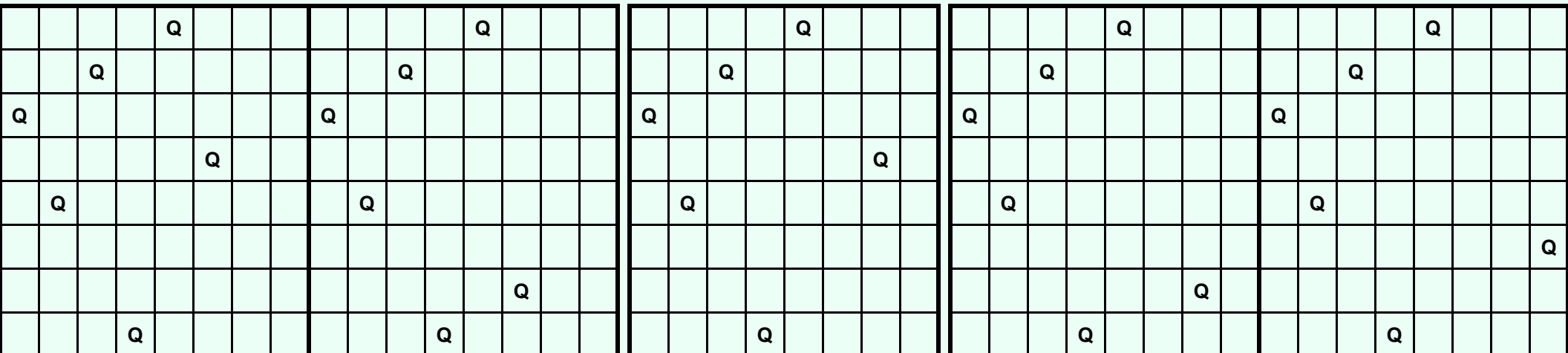
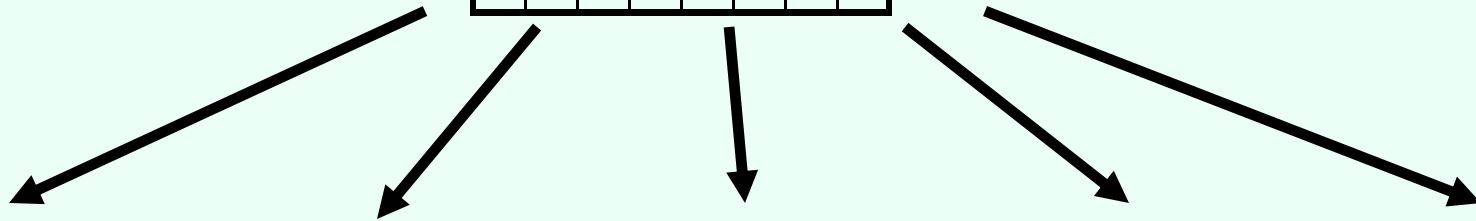
				Q			
		Q					
Q							
						Q	
	Q						
							Q
					Q		
			Q				

Search formulation of the queens puzzle

- **Successors:** all valid ways of placing additional queen on the board; **goal:** eight queens placed

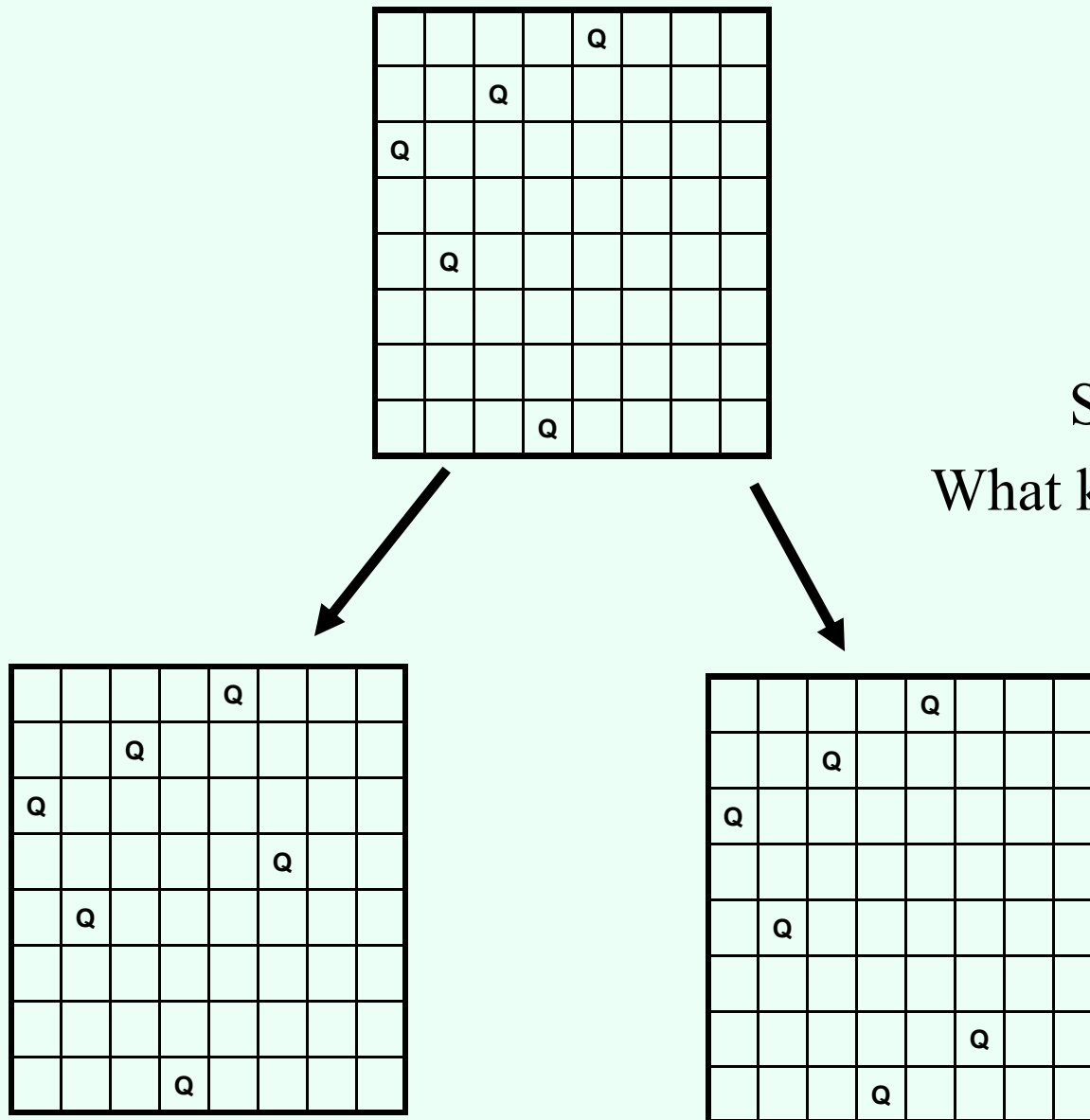


How big is this tree? How many leaves?



Search formulation of the queens puzzle

- **Successors:** all valid ways of placing a queen in the next column; **goal:** eight queens placed



Search tree size?

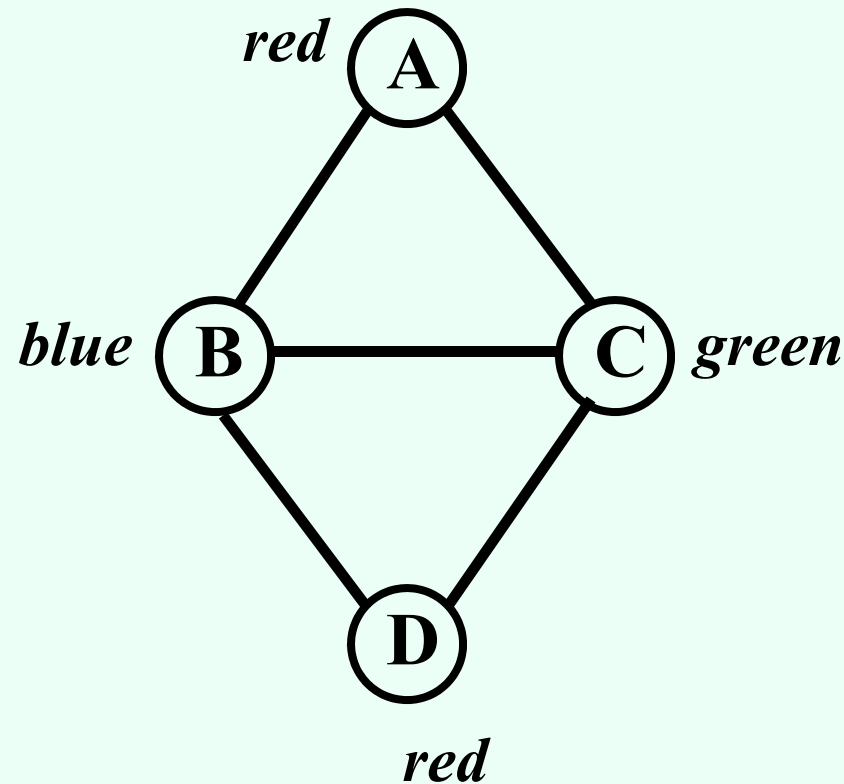
What kind of search is best?

Constraint satisfaction problems (CSPs)

- Defined by:
 - A set of **variables** x_1, x_2, \dots, x_n
 - A **domain** D_i for each variable x_i
 - **Constraints** C_1, C_2, \dots, C_m
- A constraint is specified by
 - A subset (often, two) of the variables
 - All the allowable joint assignments to those variables
- Goal: find a **complete, consistent** assignment
- Queens problem: (other examples in next slides)
 - x_i in $\{1, \dots, 8\}$ indicates in which row in the i th column to place a queen
 - For example, constraint on x_1 and x_2 : $\{(1,3), (1,4), (1,5), (1,6), (1,7), (1,8), (2,4), (2,5), \dots, (3,1), (3,5), \dots \dots\}$

Graph coloring

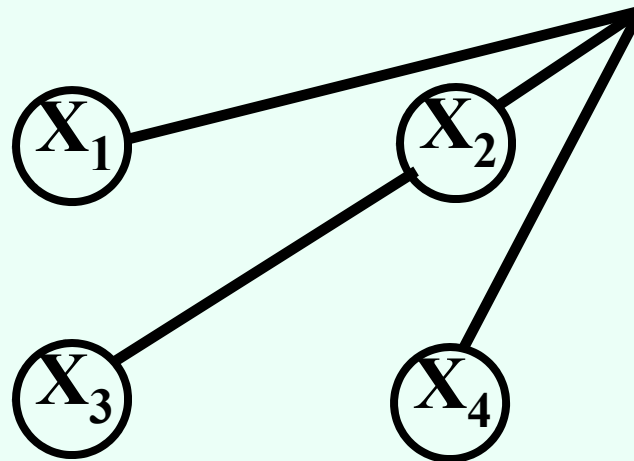
- Fixed number of colors; no two adjacent nodes can share a color



Satisfiability

- Formula in conjunctive normal form:
 $(x_1 \text{ OR } x_2 \text{ OR NOT}(x_4)) \text{ AND } (\text{NOT}(x_2) \text{ OR NOT}(x_3)) \text{ AND } \dots$
 - Label each variable x_j as true or false so that the formula becomes true

Constraint hypergraph:
each hyperedge
represents a constraint



Cryptarithmic puzzles

T W O

T W O +

F O U R

E.g., setting $F = 1$, $O = 4$, $R = 8$, $T = 7$, $W = 3$,
 $U = 6$ gives $734 + 734 = 1468$

Cryptarithmic puzzles...

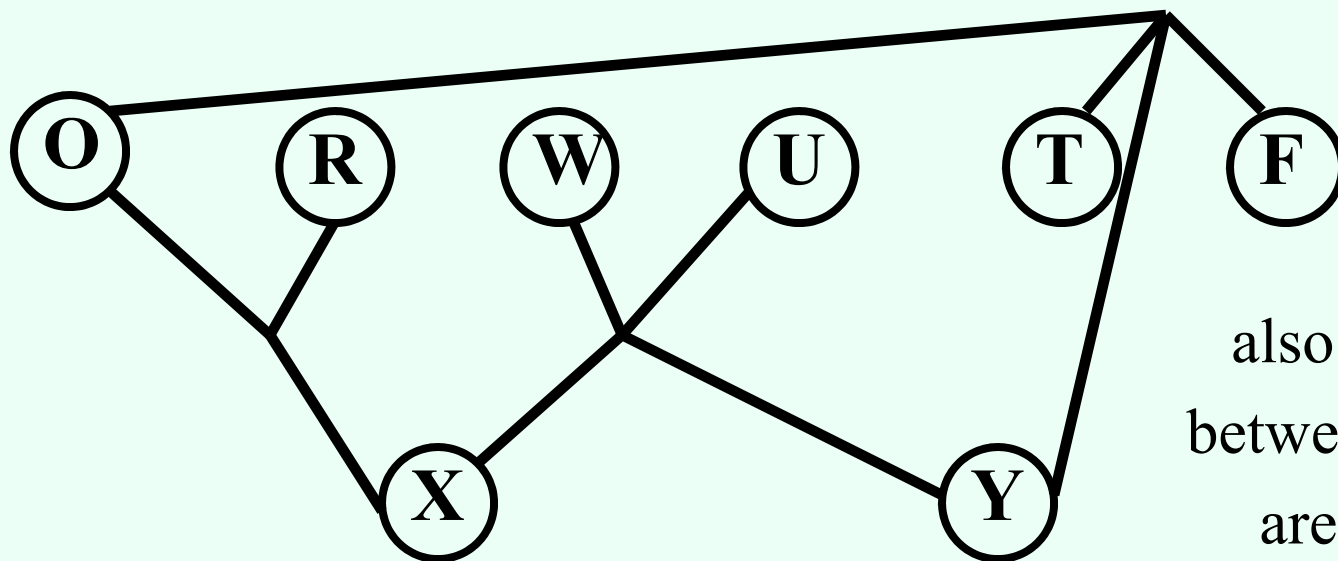
$$\begin{array}{r} T W O \\ T W O + \\ \hline F O U R \end{array}$$

Trick: introduce **auxiliary variables** X, Y

$$O + O = 10X + R$$

$$W + W + X = 10Y + U$$

$$T + T + Y = 10F + O$$



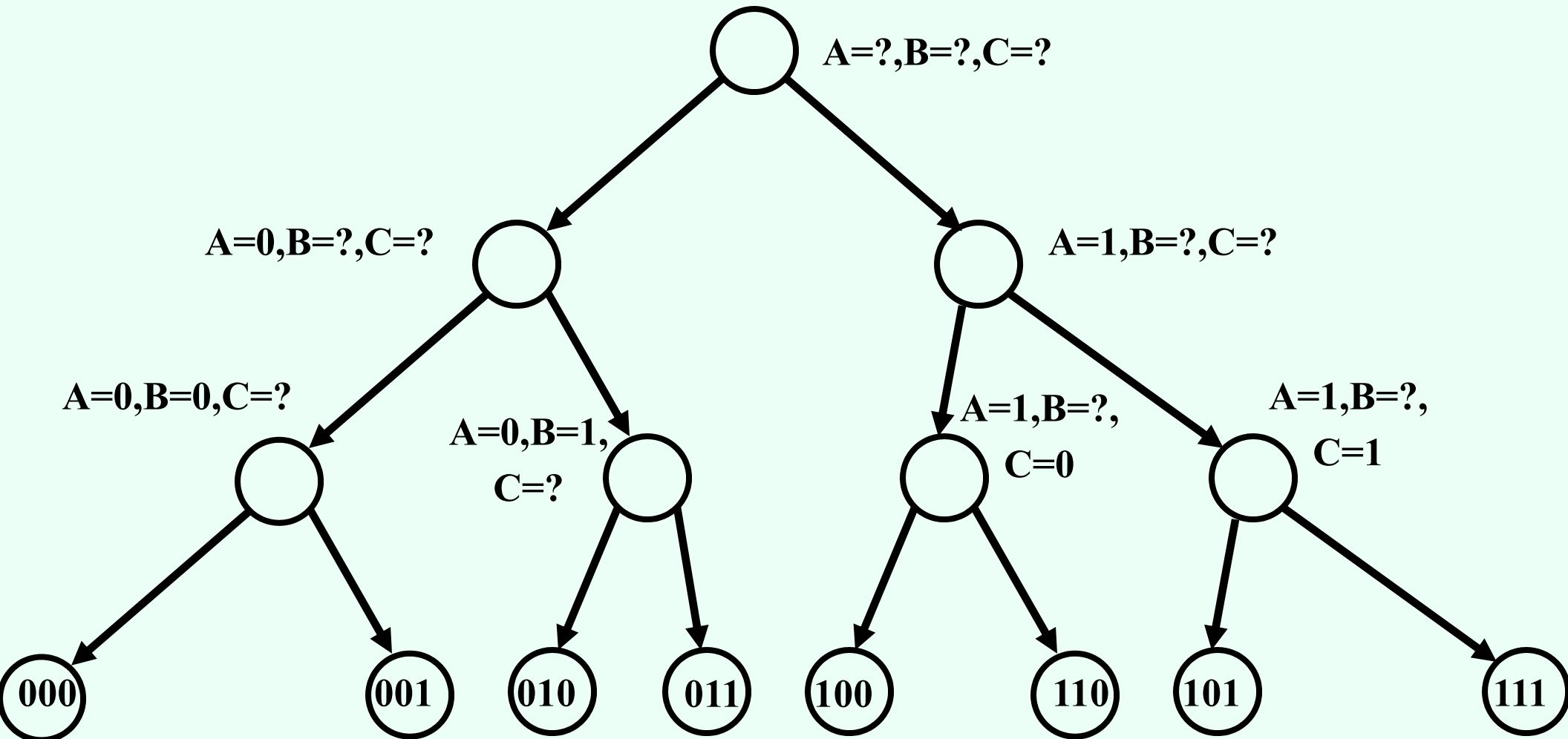
also need pairwise constraints between original variables if they are supposed to be different

Generic approaches to solving CSPs

- State: some variables assigned, others not assigned
- Naïve successors definition: any way of assigning a value to an unassigned variable results in a successor
 - Can check for consistency when expanding
 - How many leaves do we get in the worst case?
- CSPs satisfy **commutativity**: order in which actions applied does not matter
- Better idea: only consider assignments for a single variable at a time
 - How many leaves?

Choice of variable to branch on is still flexible!

- Do not always need to choose same variable at same level
- Each of variables A , B , C takes values in $\{0,1\}$



- Can you **prove** that this never increases the size of the tree?

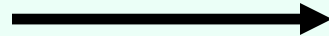
A generic recursive search algorithm

- **Search(*assignment*, *constraints*)**
- If *assignment* is complete, return it
- Choose an unassigned variable x
- For every value v in x 's domain, if setting x to v in *assignment* does not violate *constraints*:
 - Set x to v in *assignment*
 - $result := \text{Search}(\text{assignment}, \text{constraints})$
 - If $result \neq \text{failure}$ return $result$
 - Unassign x in *assignment*
- Return *failure*

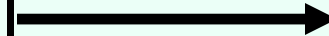
Keeping track of remaining possible values

- For every variable, keep track of which values are still possible

			Q	X	X	X
	Q			X	X	X
Q				X	X	X
						X
	Q			X	X	X
				X	X	
						X
		Q		X	X	X



				Q	X	X	
		Q			X	X	
Q					X	X	
					X		
	Q				X	X	
					X	X	Q
						X	
			Q		X	X	



				Q			
		Q					
Q							
							Q
	Q						
							Q
				Q			
			Q				

only one possibility
for last column; might
as well fill in

now only one left for
other two columns

done!
(no real branching
needed!)

- General heuristic: branch on variable with fewest values remaining

Arc consistency

- Take two variables connected by a constraint
- Is it true that for **every** remaining value d of the first variable, there exists **some** value d' of the other variable so that the constraint is satisfied?
 - If so, we say the **arc from the first to the second variable is consistent**
 - If not, can remove the value d
- General concept: **constraint propagation**

q			x				x
			x				
	q		x				x
			x				
			x	q			x
			x				

Is the arc from the fifth to the eighth column consistent?

What about the arc from the eighth to the fifth?

Maintaining arc consistency

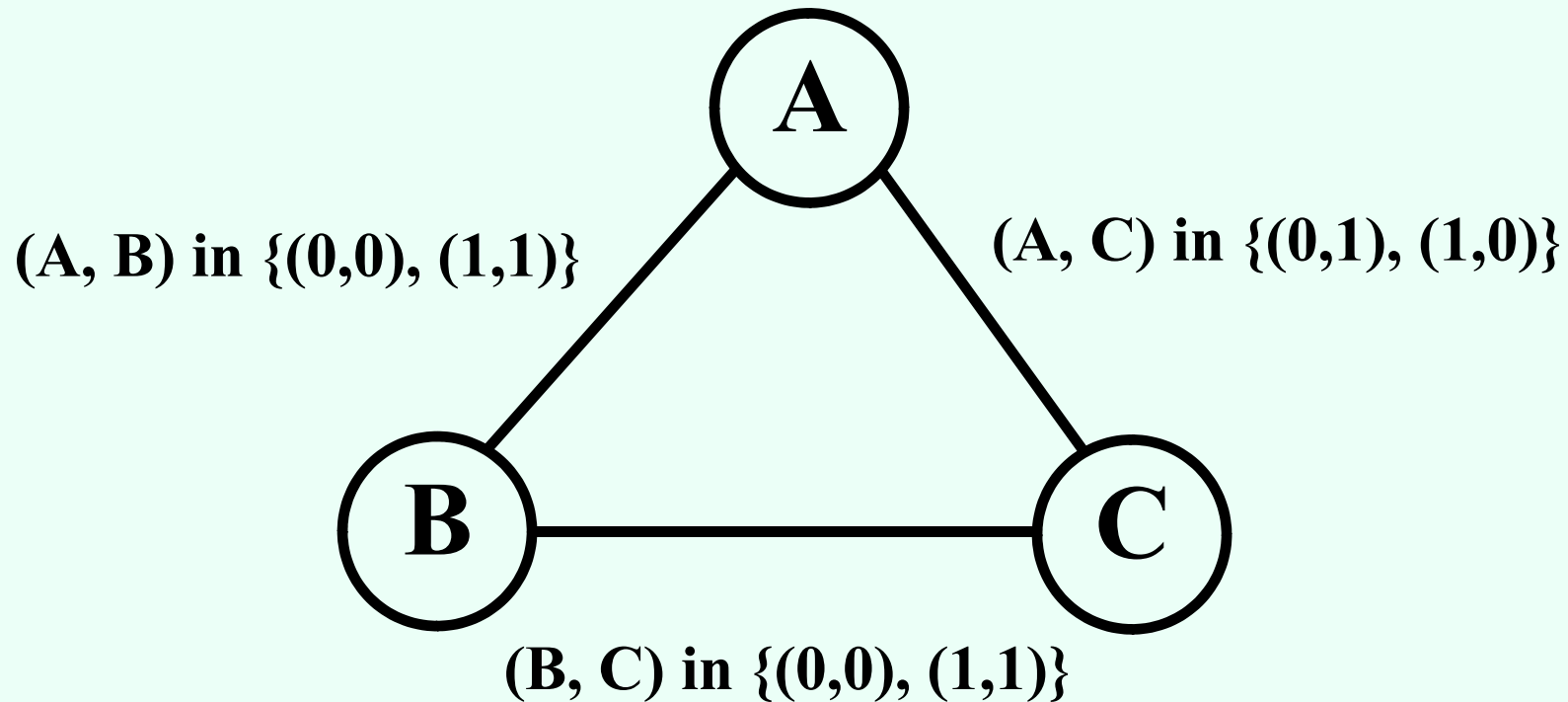
- Maintain a queue Q of all ordered pairs of variables with a constraint (arcs) that need to be checked
- Take a pair (x, y) from the queue
- For every value v in x 's domain, check if there is some value w in y 's domain so that $x=v, y=w$ is consistent
 - If not, remove v from x 's domain
- If anything was removed from x 's domain, add **every** arc (z, x) to Q
- Continue until Q is empty

- Runtime?
- n variables, d values per domain
- $O(n^2)$ arcs;
- each arc is added to the queue at most d times;
- consistency of an arc can be checked with d^2 lookups in the constraint's table;
- so $O(n^2d^3)$ lookups
- Can we do better?

Maintaining arc consistency (2)

- For every arc (x, y) , for every value v for x , maintain the number $n((x, y), v)$ of remaining values for y that are consistent with $x=v$
- Every time that some $n((x, y), v) = 0$,
 - remove v from x 's domain;
 - for every arc (z, x) , for every value w for z , if $(x=v, z=w)$ is consistent with the constraint, reduce $n((z, x), w)$ by 1
- Runtime:
 - for every arc (z, x) (n^2 of them), a value is removed from x 's domain at most d times;
 - each time we have to check for at most d of z 's values whether it is consistent with the removed value for x ;
 - so $O(n^2d^2)$ lookups

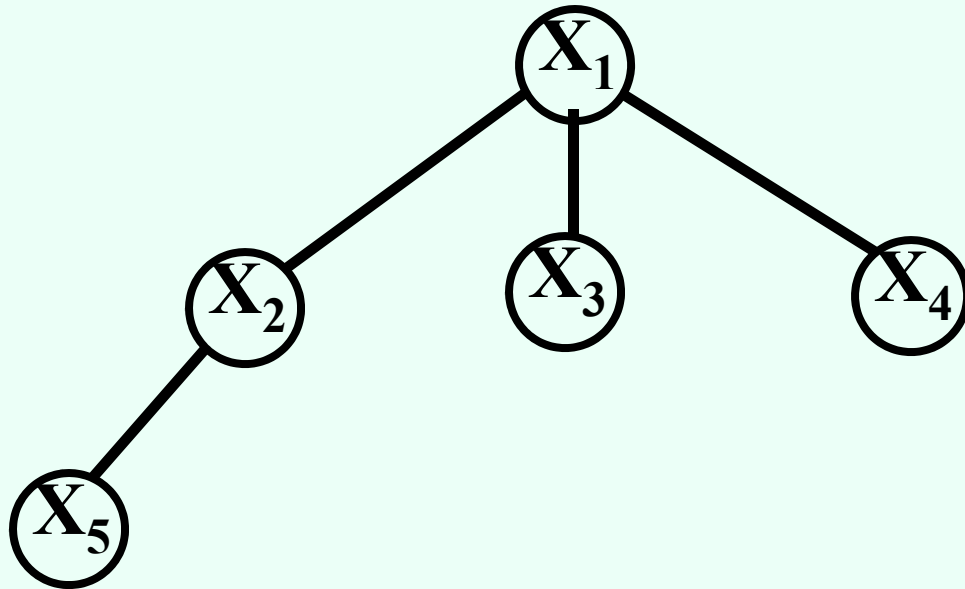
An example where arc consistency fails



- $A = B, B = C, C \neq A$ – obviously inconsistent
 - \sim Moebius band
- However, arc consistency cannot eliminate anything

Tree-structured constraint graphs

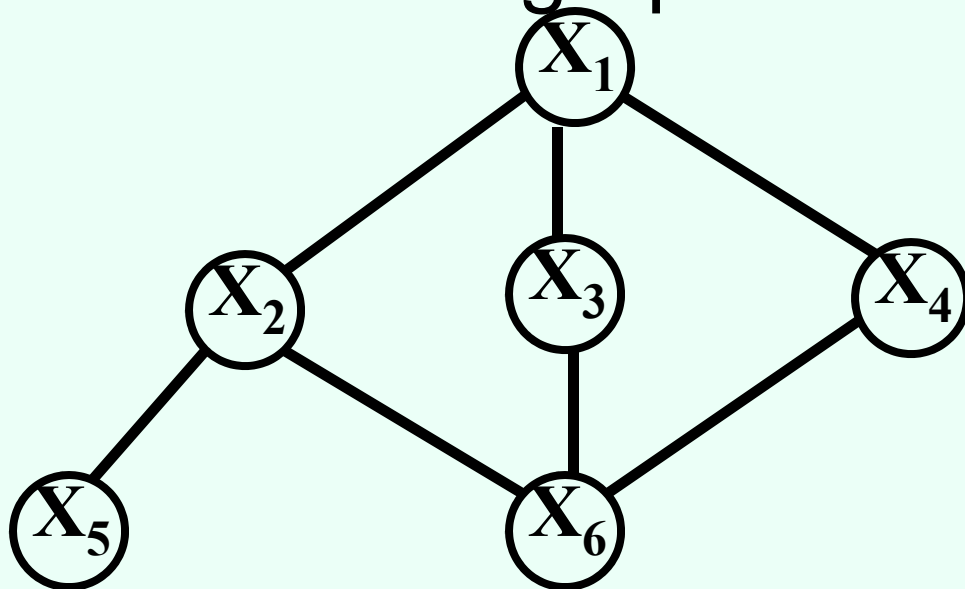
- Suppose we only have pairwise constraints and the graph is a **tree** (or **forest** = multiple disjoint trees)



- Dynamic program for solving this (linear in #variables):
 - Starting from the leaves and going up, for each node x , compute all the values for x such that the subtree rooted at x can be solved
 - Equivalently: apply arc consistency from each parent to its children, starting from the bottom
 - If no domain becomes empty, once we reach the top, easy to fill in solution

Generalizations of the tree-based approach

- What if our constraint graph is “almost” a tree?



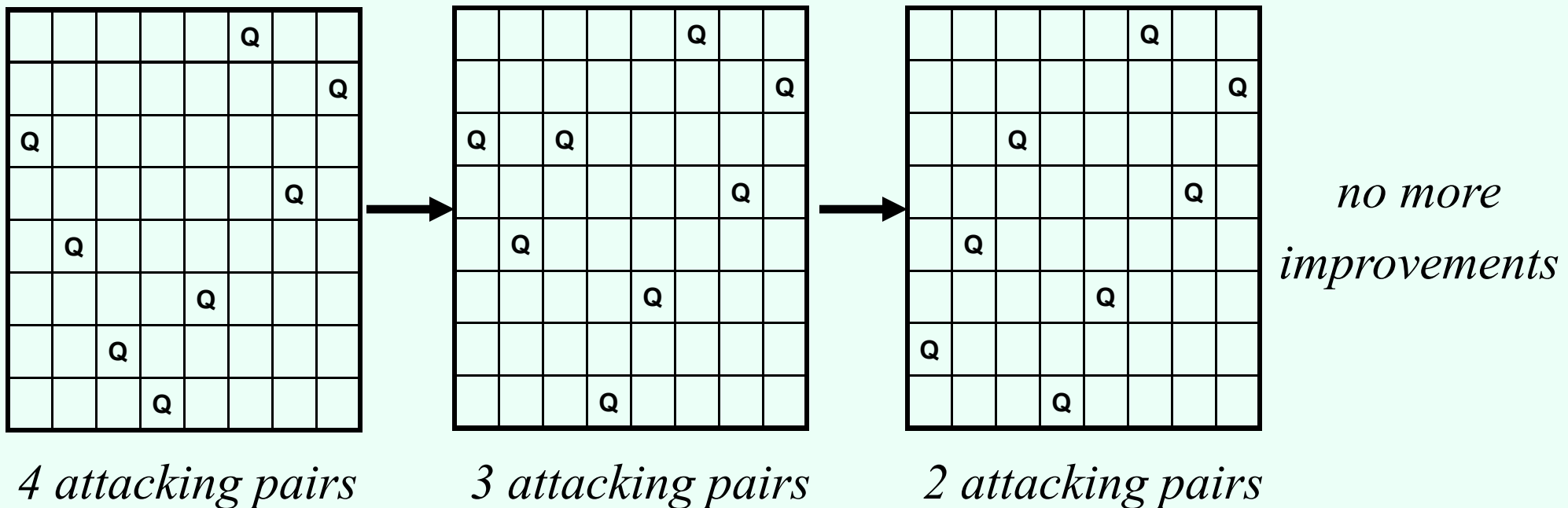
- A **cycle cutset** is a set of variables whose removal results in a tree (or forest)
 - E.g. $\{X_1\}$, $\{X_6\}$, $\{X_2, X_3\}$, $\{X_2, X_4\}$, $\{X_3, X_4\}$
- Simple algorithm: for every internally consistent assignment to the cutset, solve the remaining tree as before (runtime?)
- Graphs of **bounded treewidth** can also be solved in polynomial time (won't define these here)

A different approach: optimization

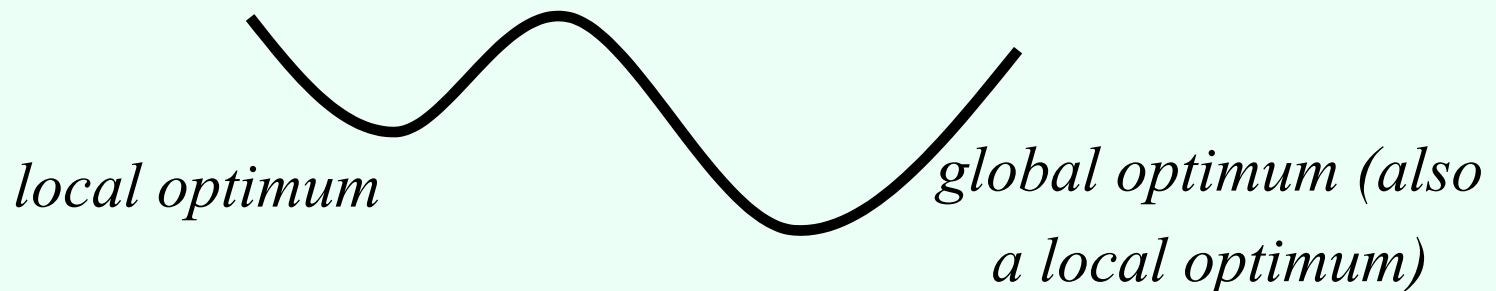
- Let's say every way of placing 8 queens on a board, one per column, is **feasible**
- Now we introduce an **objective**: minimize the number of pairs of queens that attack each other
 - More generally, minimize the number of violated constraints
- Pure optimization

Local search: hill climbing

- Start with a complete state
- Move to successor with best (or at least better) objective value
 - Successor: move one queen within its column



- Local search can get stuck in a **local optimum**



Avoiding getting stuck with local search

- **Random restarts:** if your hill-climbing search fails (or returns a result that may not be optimal), restart at a random point in the search space
 - Not always easy to generate a random state
 - Will **eventually** succeed (why?)
- **Simulated annealing:**
 - Generate a random successor (possibly worse than current state)
 - Move to that successor with some probability that is sharply decreasing in the badness of the state
 - Also, over time, as the “temperature decreases,” probability of bad moves goes down

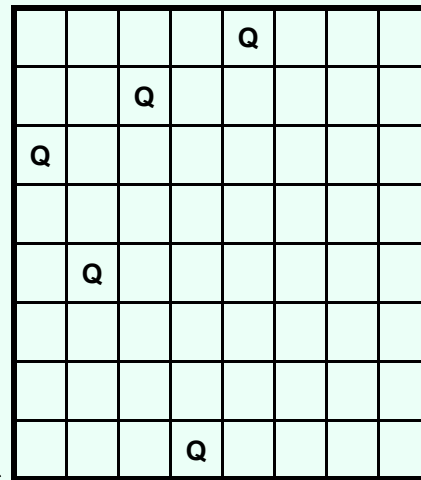
Constraint optimization

- Like a CSP, but with an objective
 - E.g., minimize number of violated constraints
 - Another example: no two queens can be in the same row or column (hard constraint), minimize number of pairs of queens attacking each other diagonally (objective)
- Can use all our techniques from before: heuristics, A^* , IDA^* , ...
- Also popular: **depth-first branch-and-bound**
 - Like depth-first search, except do not stop when first feasible solution found; keep track of best solution so far
 - Given admissible heuristic, do not need to explore nodes that are worse than best solution found so far

Minimize #violated diagonal constraints

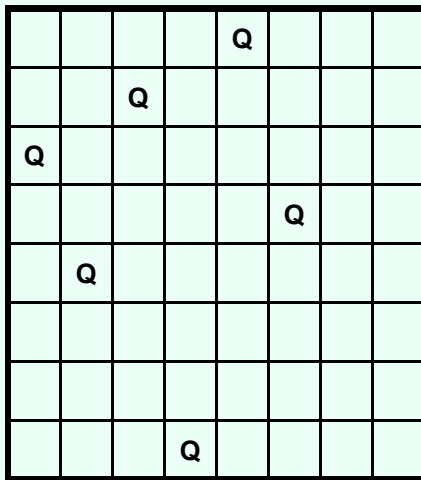
- **Cost of a node:** #violated diagonal constraints so far

- No heuristic
*(matter of definition;
could just as well say
that violated
constraints so far is
the heuristic and
interior nodes have
no cost)*

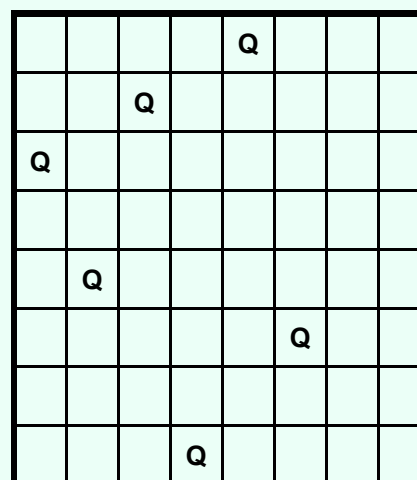


cost = 0

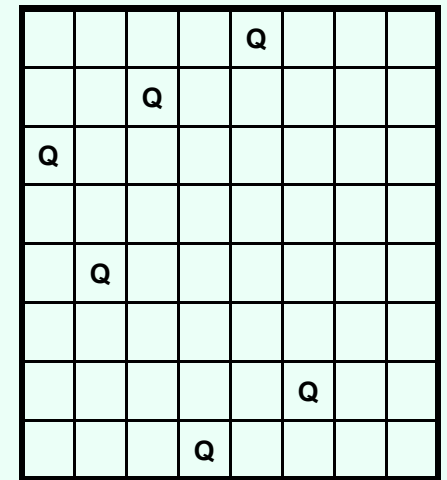
cost = 0



cost = 1



cost = 0



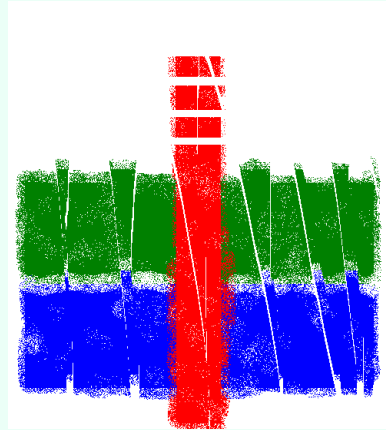
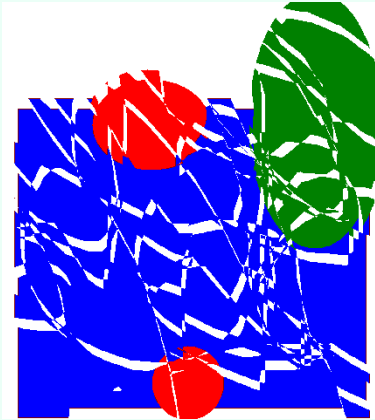
Depth first branch and bound will find a suboptimal solution here first (no way to tell at this point this is worse than right node)

A* (=uniform cost here), IDA* (=iterative lengthening here) will **never** explore this node

Optimal solution is down here (cost 0)

Linear programs: example

- We make reproductions of two paintings



maximize $3x + 2y$

subject to

$$4x + 2y \leq 16$$

$$x + 2y \leq 8$$

$$x + y \leq 5$$

$$x \geq 0$$

$$y \geq 0$$

- Painting 1 sells for \$30, painting 2 sells for \$20
- Painting 1 requires 4 units of blue, 1 green, 1 red
- Painting 2 requires 2 blue, 2 green, 1 red
- We have 16 units blue, 8 green, 5 red

Solving the linear program graphically

maximize $3x + 2y$

subject to

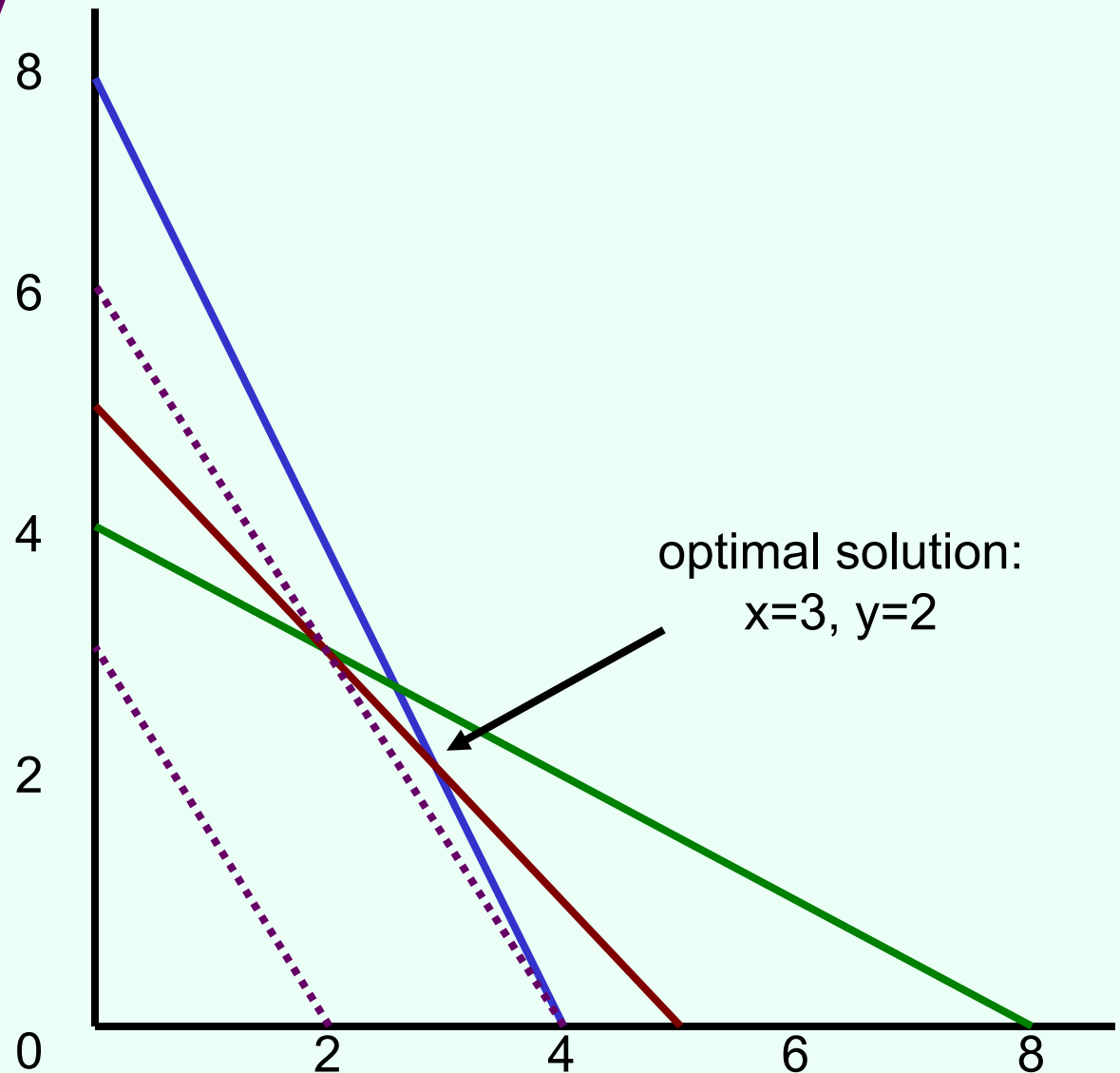
$$4x + 2y \leq 16$$

$$x + 2y \leq 8$$

$$x + y \leq 5$$

$$x \geq 0$$

$$y \geq 0$$



Modified LP

maximize $3x + 2y$

subject to

$$4x + 2y \leq 15$$

$$x + 2y \leq 8$$

$$x + y \leq 5$$

$$x \geq 0$$

$$y \geq 0$$

Optimal solution: $x = 2.5$,
 $y = 2.5$

Solution value = $7.5 + 5 =$
 12.5

Half paintings?

Integer (linear) program

maximize $3x + 2y$

subject to

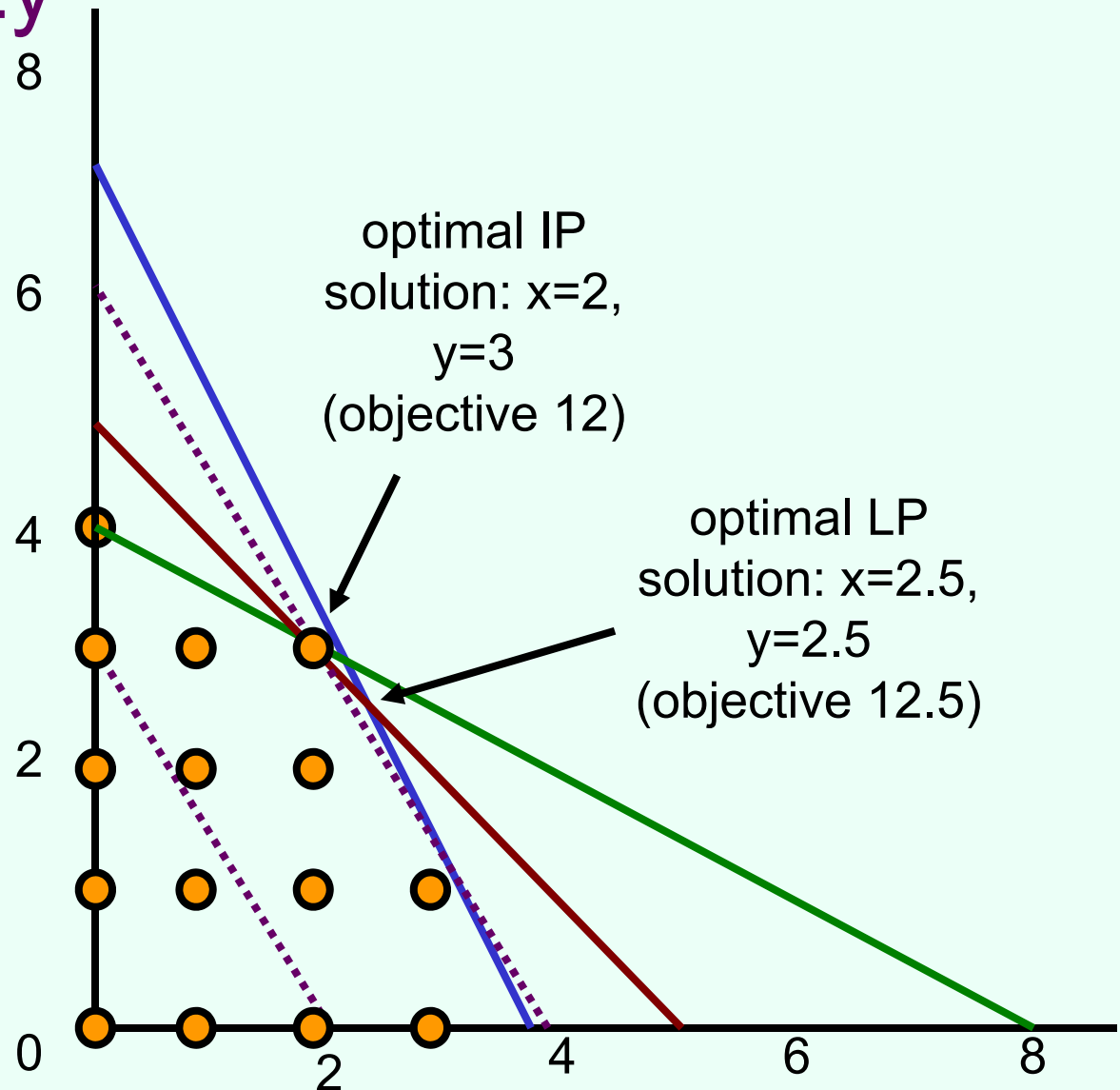
$$4x + 2y \leq 15$$

$$x + 2y \leq 8$$

$$x + y \leq 5$$

$x \geq 0$, integer

$y \geq 0$, integer



Mixed integer (linear) program

maximize $3x + 2y$

subject to

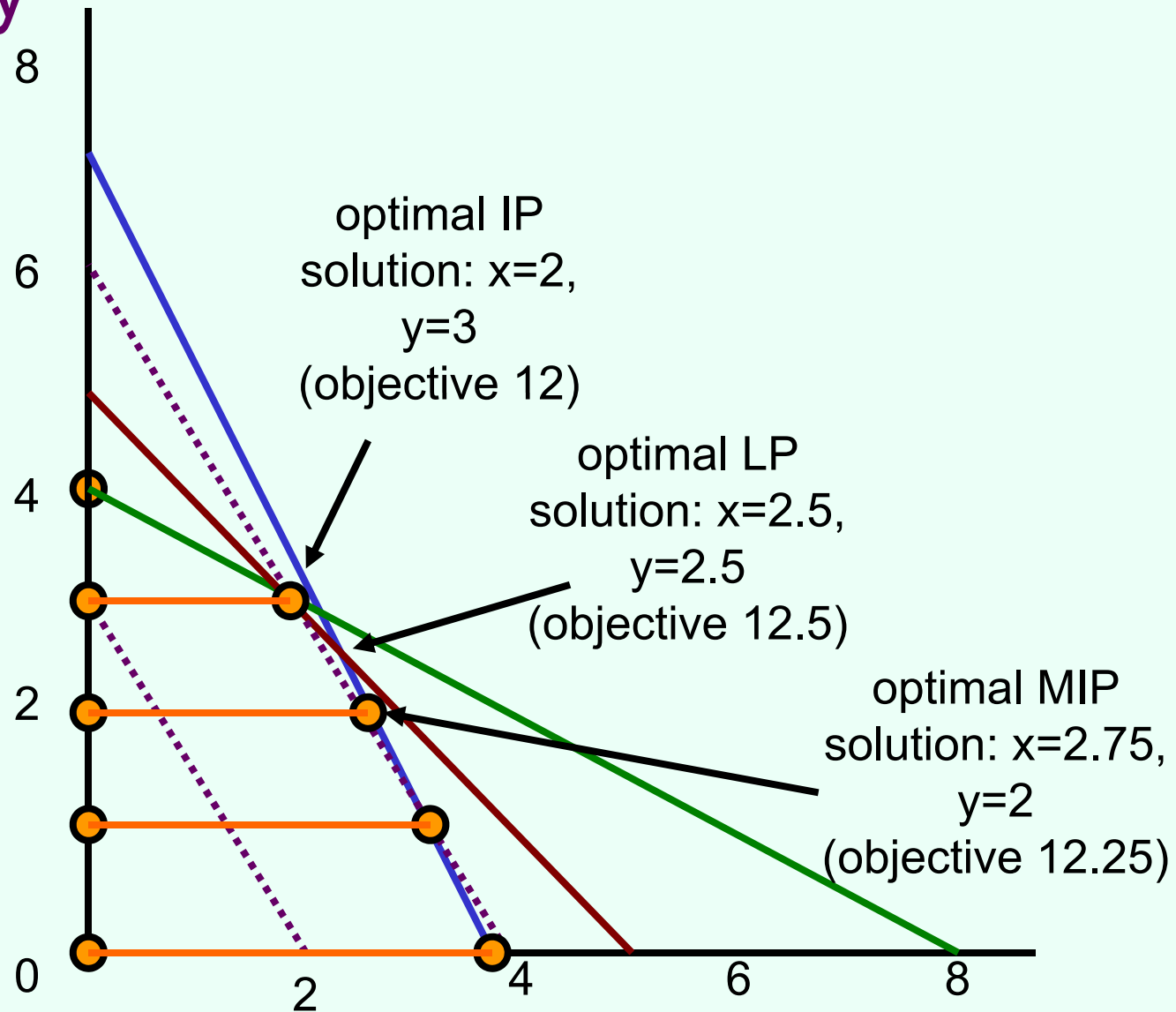
$$4x + 2y \leq 15$$

$$x + 2y \leq 8$$

$$x + y \leq 5$$

$$x \geq 0$$

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



Solving linear/integer programs

- Linear programs can be solved efficiently
 - Simplex, ellipsoid, interior point methods...
- (Mixed) integer programs are NP-hard to solve
 - Quite easy to model many standard NP-complete problems as integer programs (try it!)
 - Search type algorithms such as branch and bound
- Standard packages for solving these
 - GNU Linear Programming Kit, CPLEX, ...
- **LP relaxation** of (M)IP: remove integrality constraints
 - Gives upper bound on MIP (~admissible heuristic)

Satisfiability as an integer program

$(x_1 \text{ OR } x_2 \text{ OR NOT}(x_4)) \text{ AND } (\text{NOT}(x_2) \text{ OR NOT}(x_3)) \text{ AND}$

...

becomes

for all x_j , $0 \leq x_j \leq 1$, x_j integer (shorthand: x_j in $\{0,1\}$)

$$x_1 + x_2 + (1-x_4) \geq 1$$

$$(1-x_2) + (1-x_3) \geq 1$$

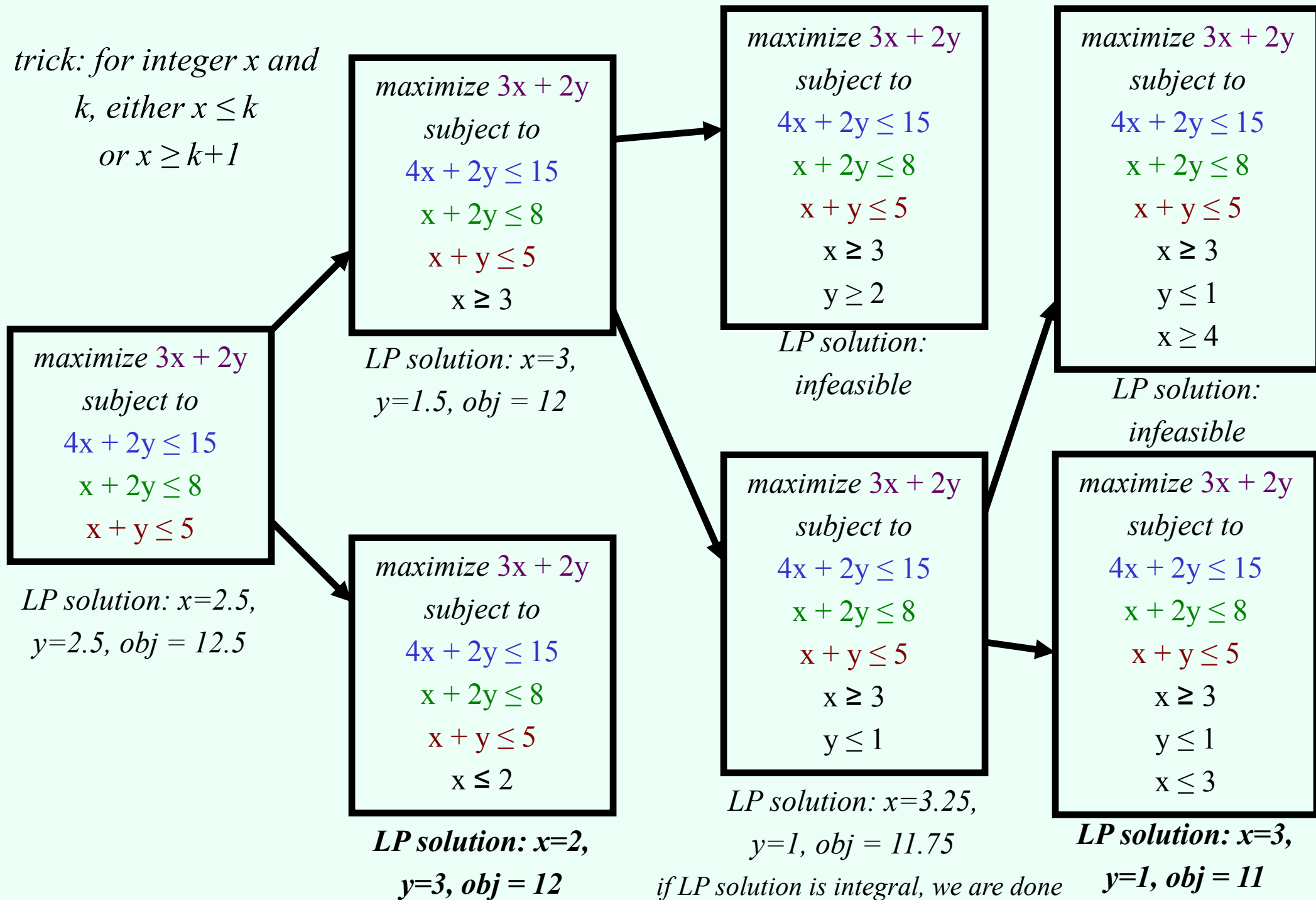
...

Solving integer programs is at least as hard as satisfiability, hence NP-hard (we have **reduced** SAT to IP)

Try modeling other NP-hard problems as (M)IP!

Solving the integer program with DFS branch and bound

trick: for integer x and k , either $x \leq k$ or $x \geq k+1$



Again with a more fortunate choice

maximize $3x + 2y$
subject to
 $4x + 2y \leq 15$
 $x + 2y \leq 8$
 $x + y \leq 5$

LP solution: $x=2.5,$
 $y=2.5, \text{obj} = 12.5$

maximize $3x + 2y$
subject to
 $4x + 2y \leq 15$
 $x + 2y \leq 8$
 $x + y \leq 5$
 $x \geq 3$

LP solution: $x=3,$
 $y=1.5, \text{obj} = 12$

maximize $3x + 2y$
subject to
 $4x + 2y \leq 15$
 $x + 2y \leq 8$
 $x + y \leq 5$
 $x \leq 2$

LP solution: $x=2,$
 $y=3, \text{obj} = 12$

done!