人工智能技术与应用

搜索II

2019.03.12

大纲

- 搜索：路径无关
- 约束满足问题CSP
- 约束优化问题COP
- AI在金融市场应用

建模-推理-学习

- 模型
  - 状态空间模型
- 推理
  - 最短路算法
- 学习
  - 感知机学习

Modeling

Inference

Learning

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

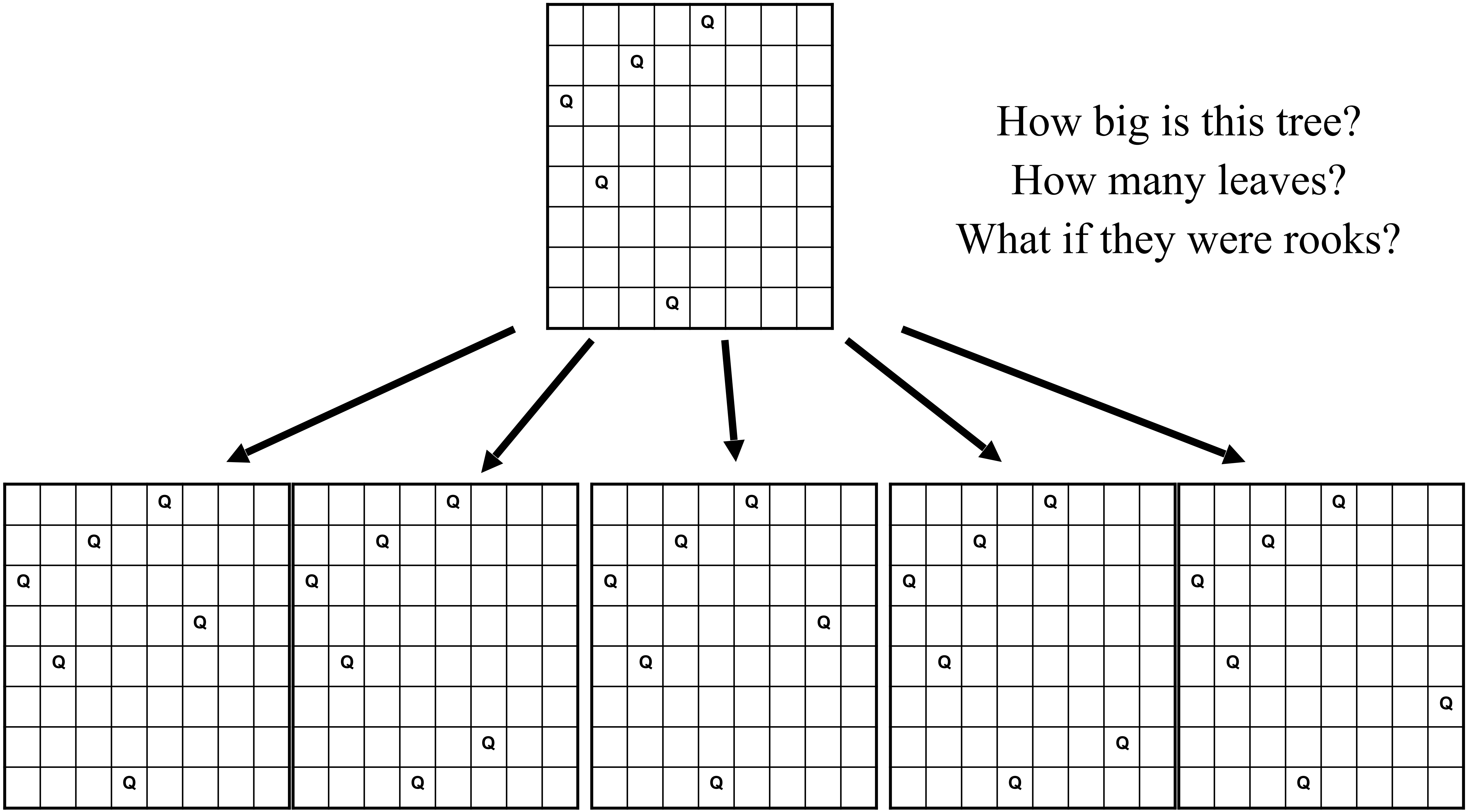
Queens puzzle

- 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



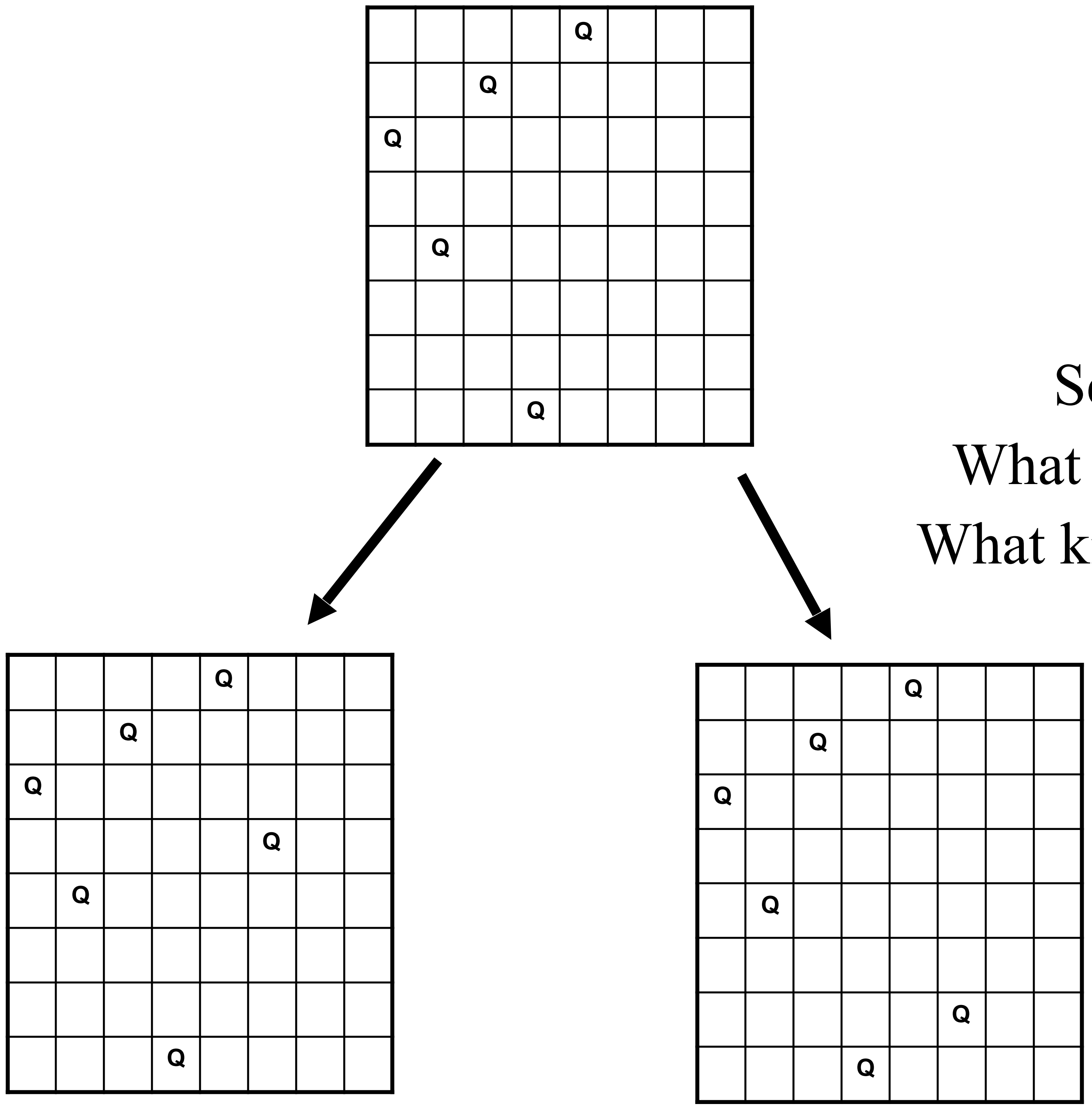
How big is this tree?

How many leaves?

What if they were rooks?

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 if they were rooks?

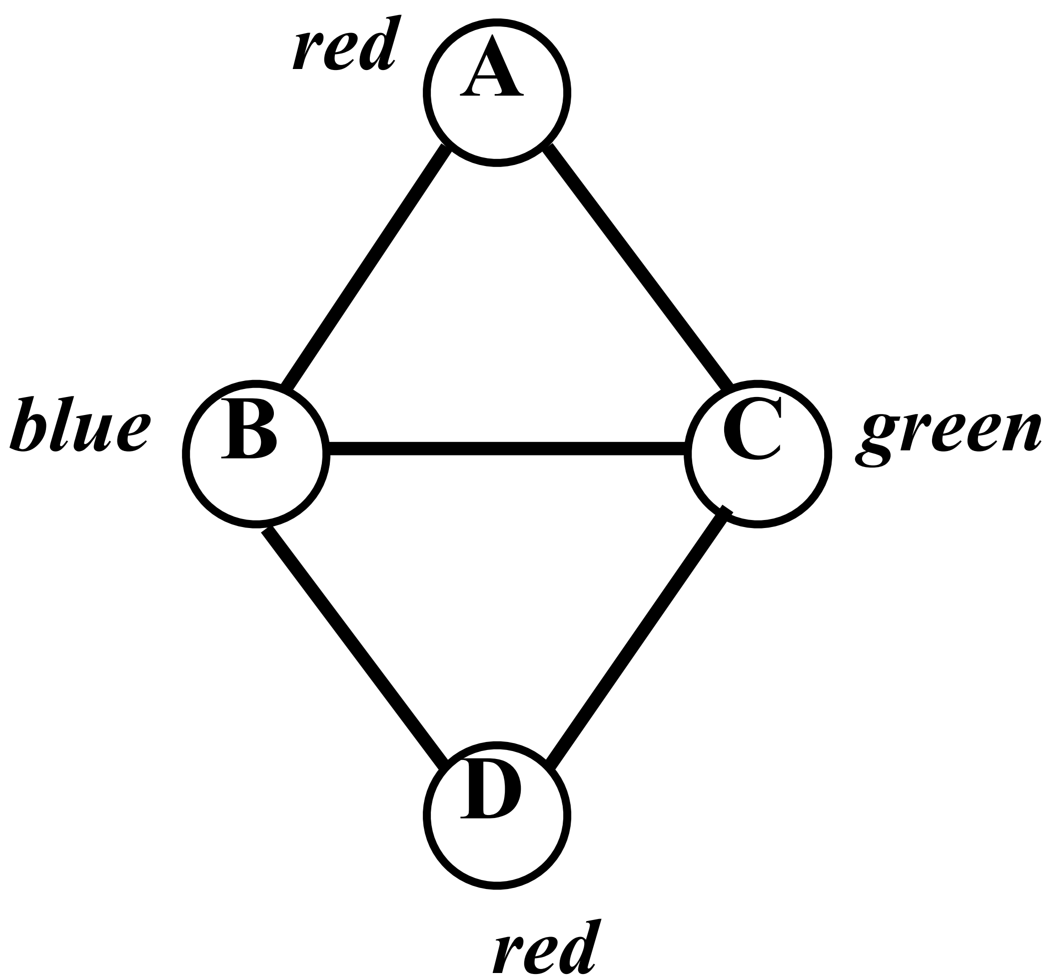
What kind of search is best?

Constraint satisfaction problems (CSPs)

- Defined by:
  - A set of variables x_1, x_2, …, x_n
  - A domain D_i for each variable x_i
  - Constraints c_1, c_2, …, 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, …, 8} indicates in which row in the ith 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), …, (3,1), (3,5), … …}

Graph coloring

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



Satisfiability

- Formula in conjunctive normal form:

  (x_1 OR x_2 OR NOT(x_4)) AND (NOT(x_2) OR NOT(x_3)) AND …

  – Label each variable x_j as true or false so that the formula becomes true

Constraint hypergraph: each hyperedge represents a constraint

Cryptarithmetic puzzles

$$\begin{array}{r} TWO \\ TWO + \\ \hline FOUR \end{array}$$

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

Cryptarithmetic puzzles…

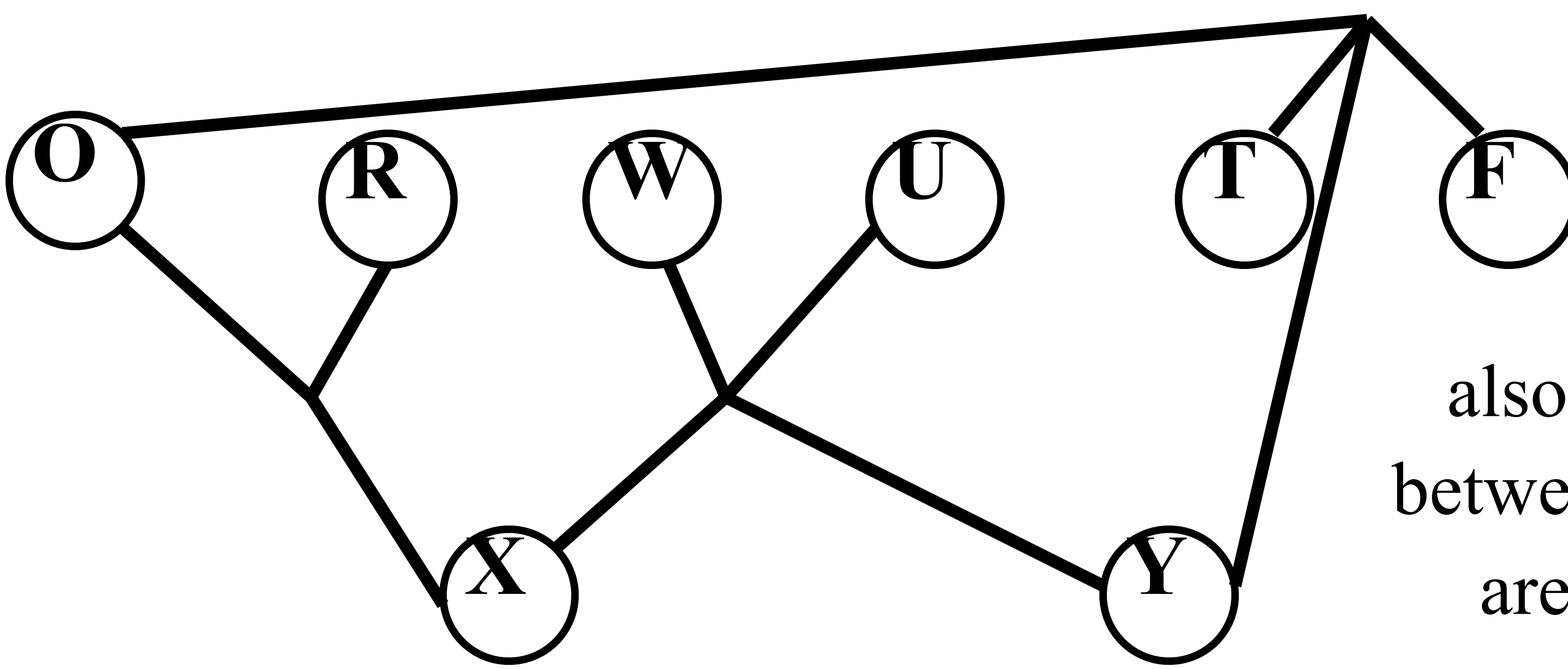
$$\begin{array}{r} T\ W\ O \\ T\ W\ O\ + \\ \hline F\ O\ U\ R \end{array}$$

Trick: introduce auxiliary variables X, Y

$O + O = 10X + R$

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

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



What would the search tree look like?

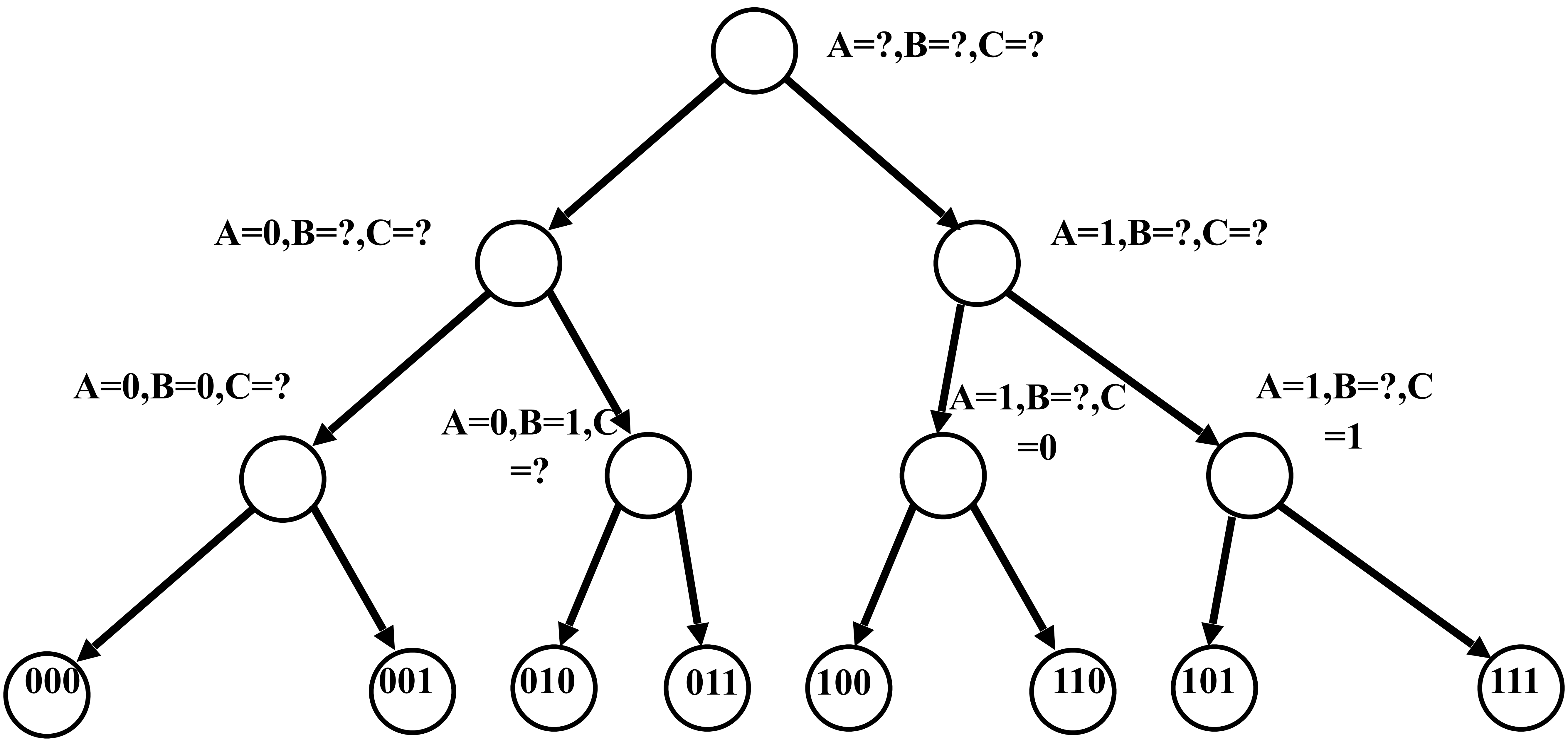
also need pairwise constraints between original variables if they are supposed to be different

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?

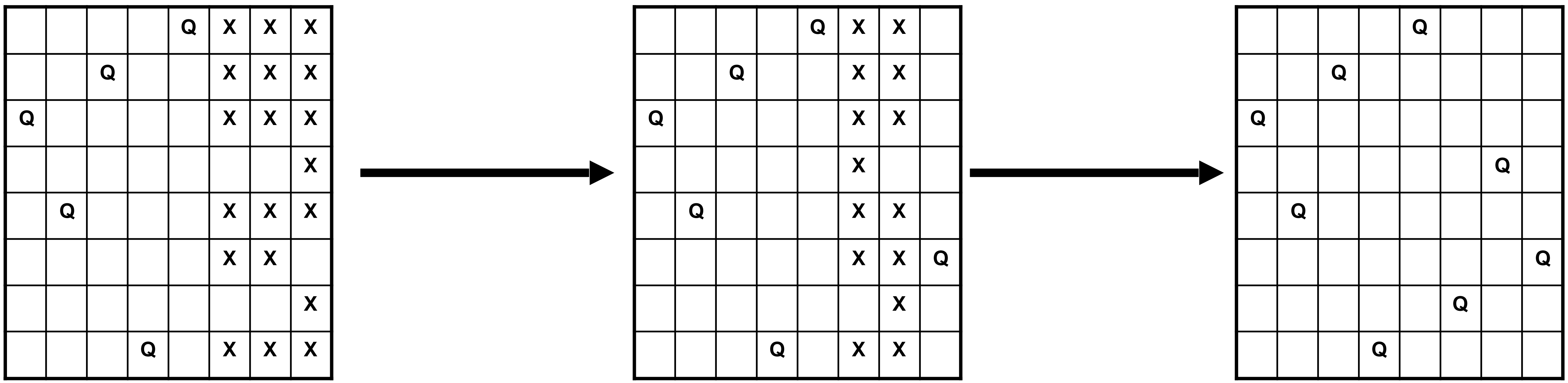
A generic recursive search algorithm

(*assignment* is a partial assignment)

- **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* := Search(*assignment*, *constraints*)
  - If *result* != *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



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

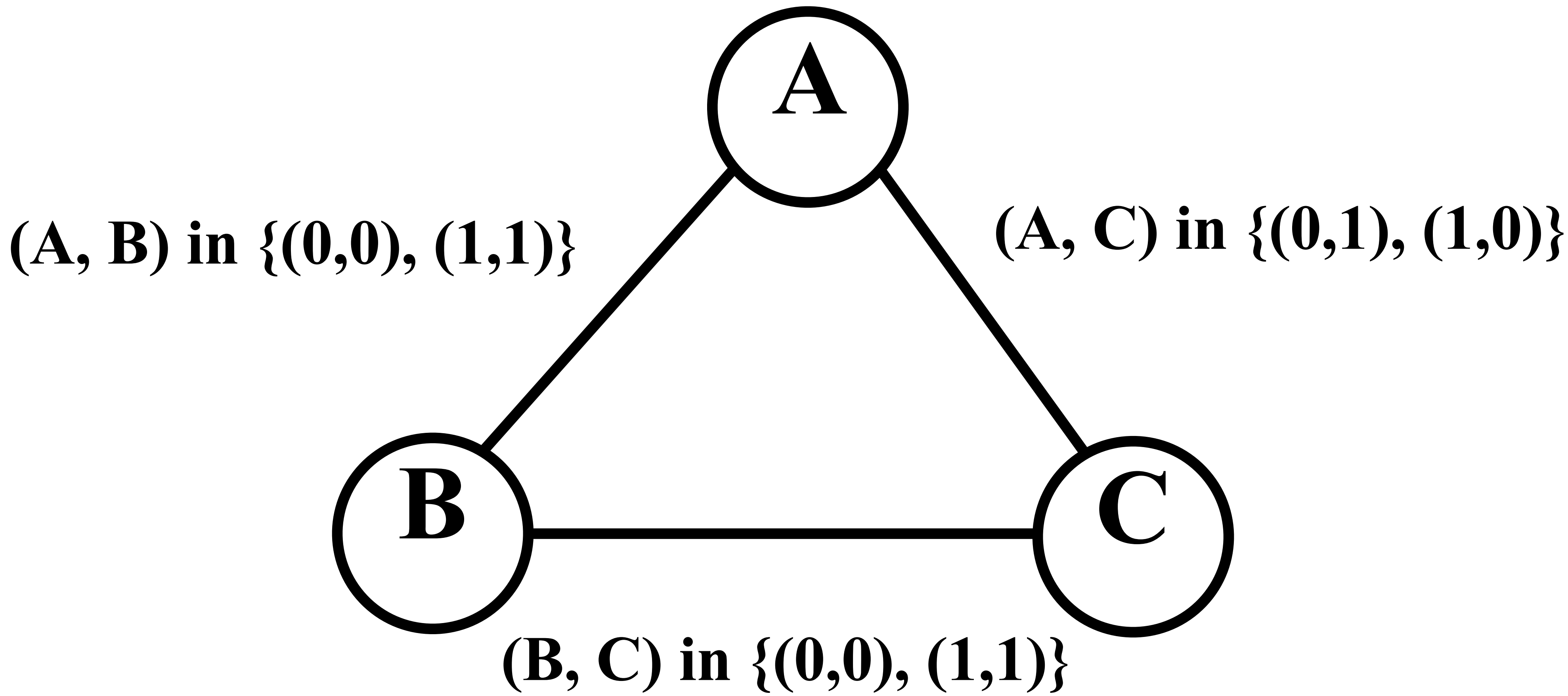
Q				X			X
				X			
		Q		X			X
				X			
				X	Q		X
				X			

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

What about the arc from the eighth to the fifth?

Consider cryptarithmetic puzzle again…

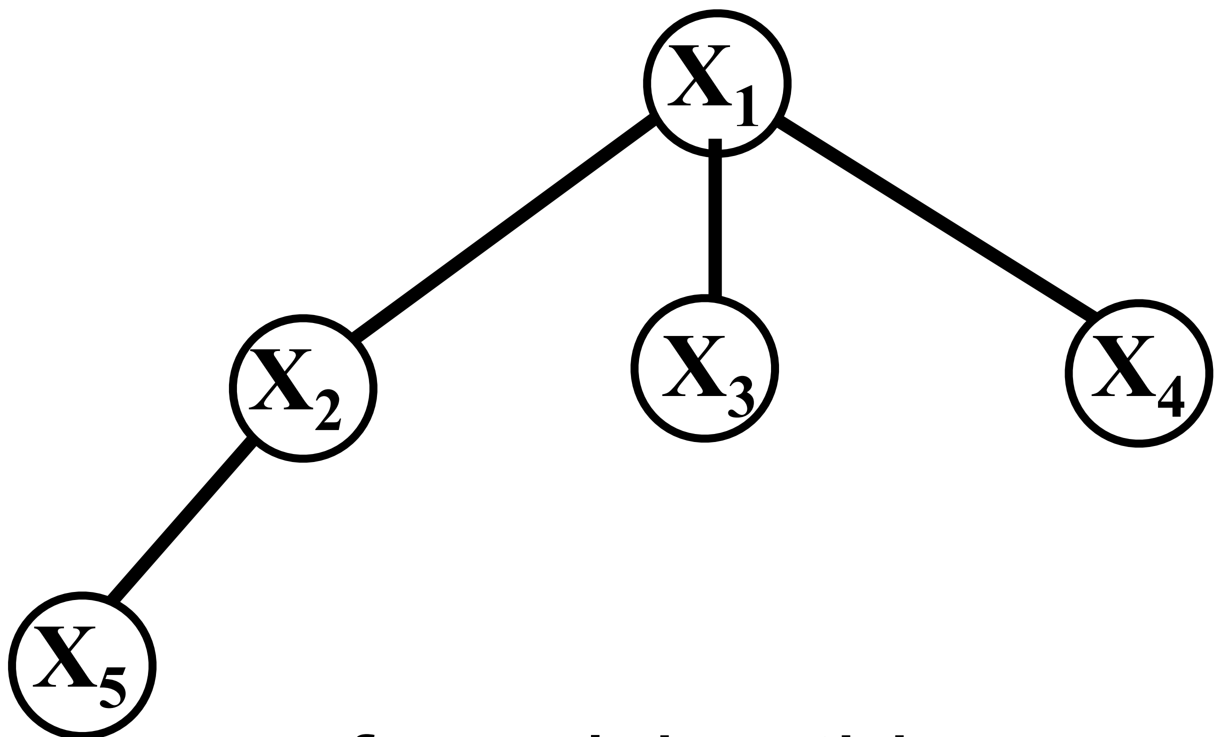
An example where arc consistency fails



- A = B, B = C, C ≠ A – obviously inconsistent
  - ~ Moebius band
- However, arc consistency cannot eliminate anything

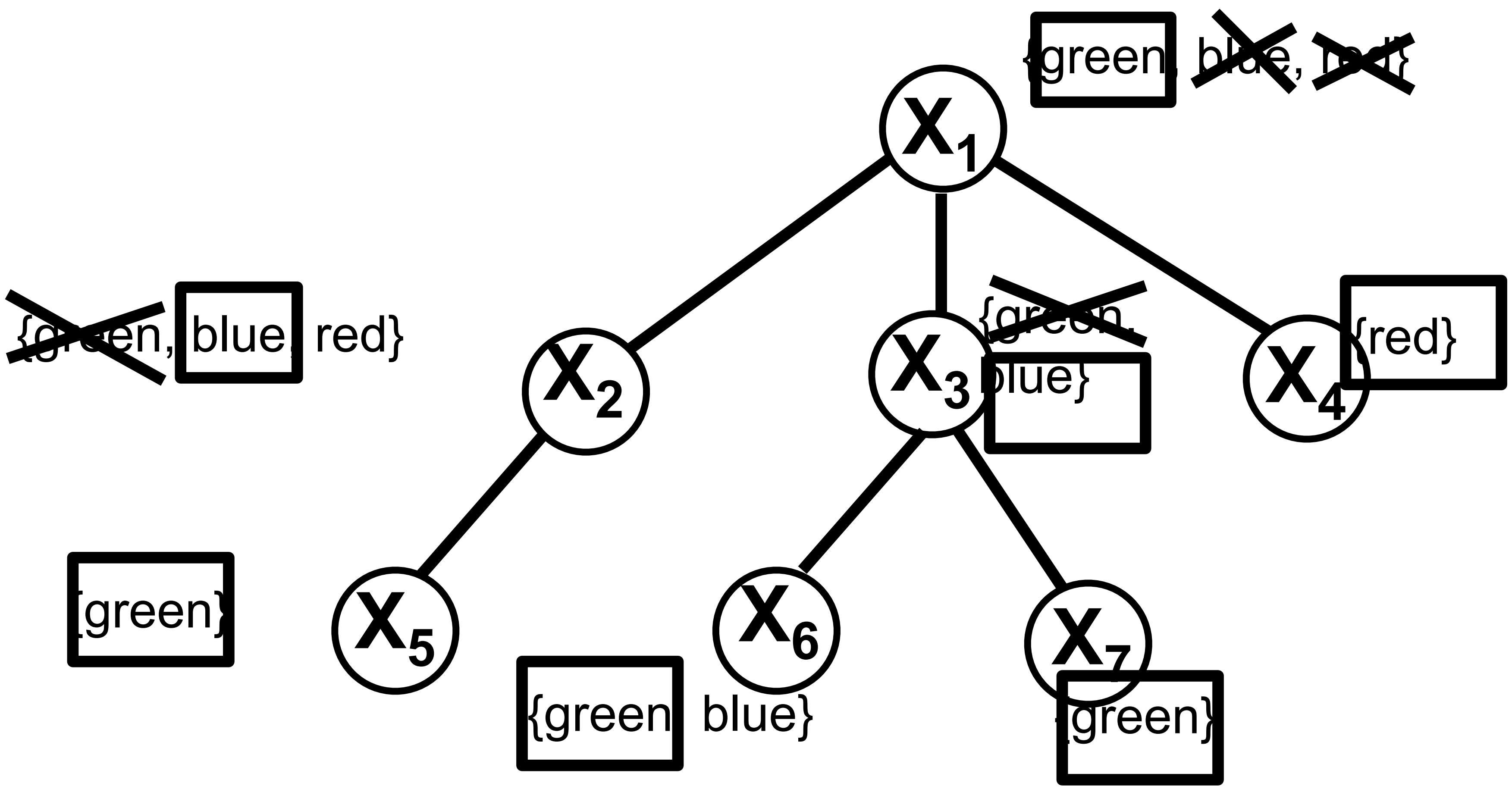
Tree-structured constraint graphs

- 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

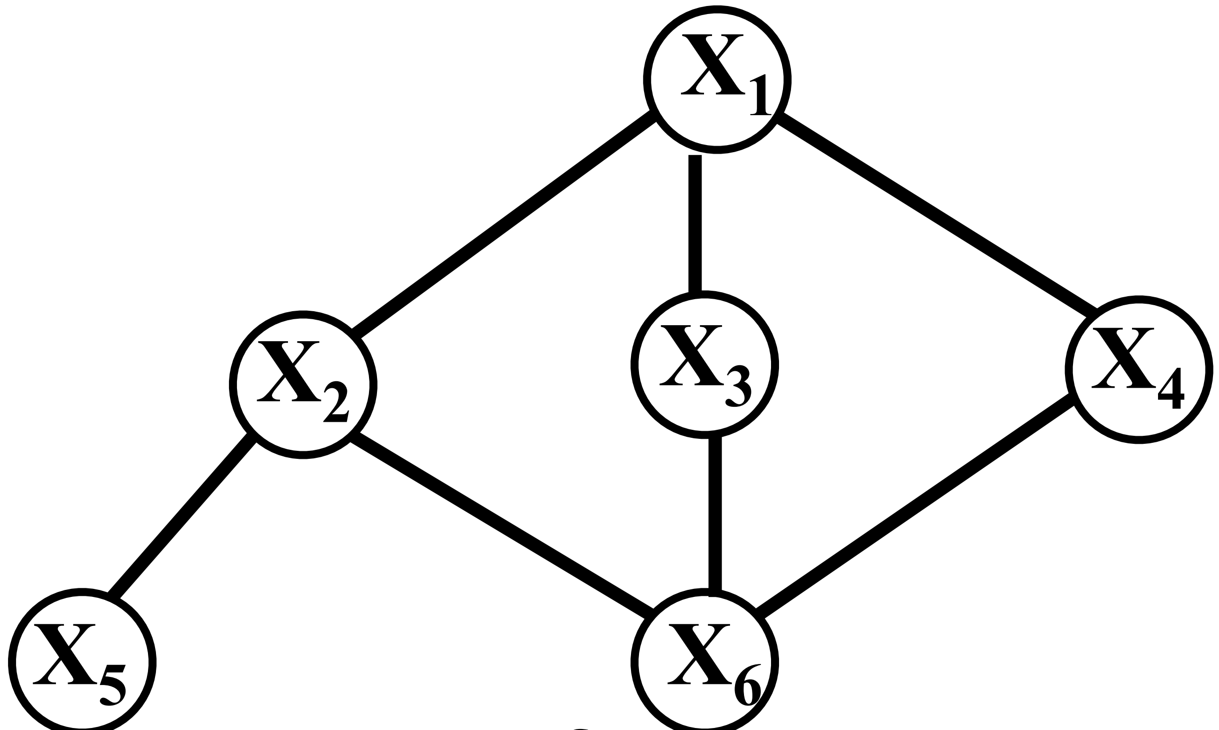
Example: graph coloring with limited set of colors per node



- Stage 1: moving upward, cross out the values that cannot work with the subtree below that node
- Stage 2: if a value remains at the root, there is a solution: go downward to pick a solution

Generalizations of the tree-based approach

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



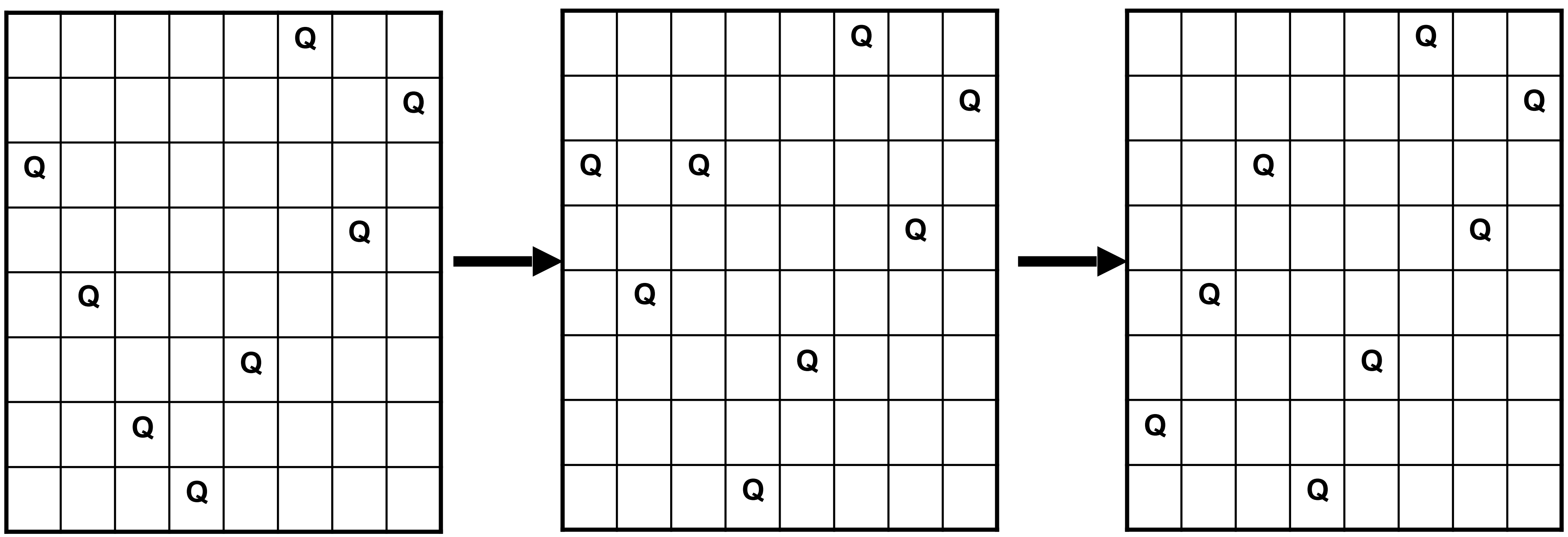
- A cycle cutset is a set of variables whose removal results in a tree (or forest)
  - E.g. $\{X_1\}$, $\{X_6\}$, $\{X_2, X_3\}$, $\{X_2, X_4\}$, $\{X_3, X_4\}$
- Simple algorithm: for every internally consistent assignment to the cutset, solve the remaining tree as before (runtime?)
- Graphs of bounded 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

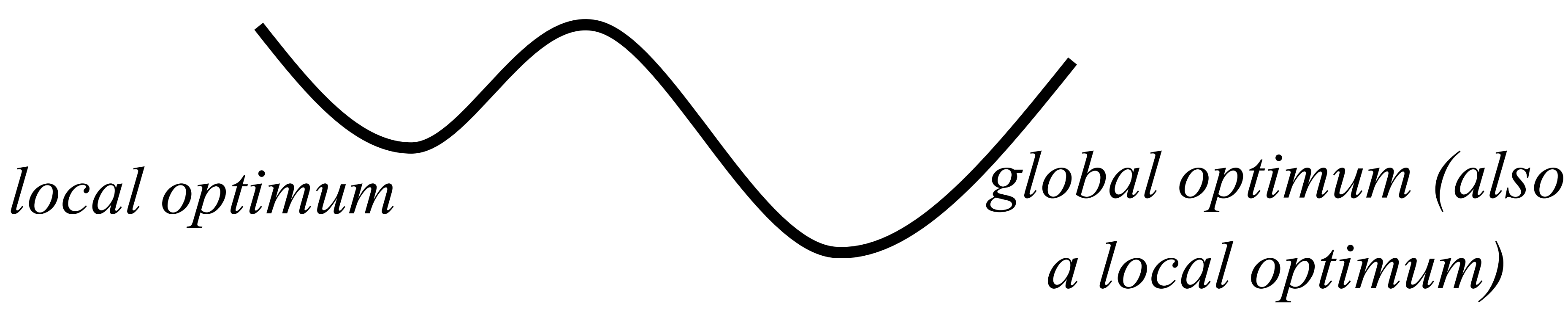
- Start with a complete state
- Move to successor with best (or at least better) objective value
  - Successor: move one queen within its column



4 attacking pairs *3 attacking pairs* *2 attacking pairs*

no more improvements

- 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

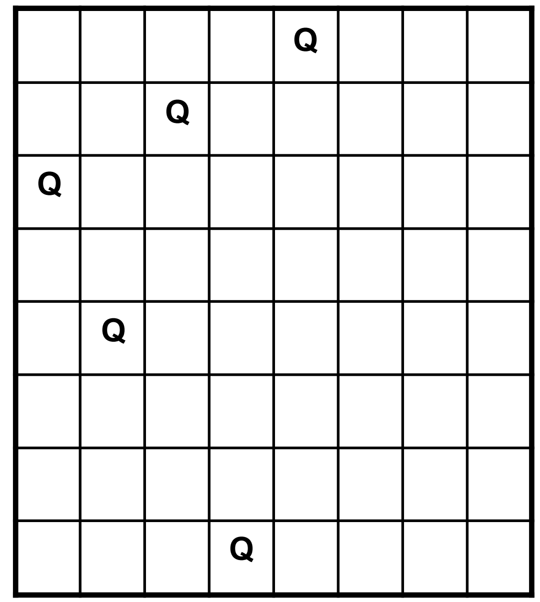
Constraint optimization

- 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

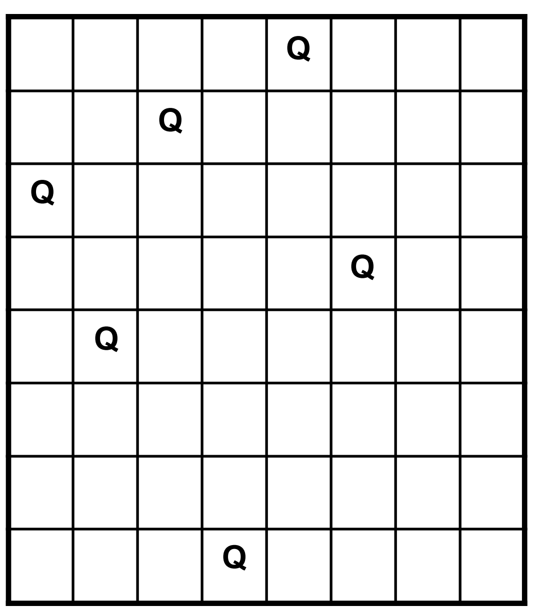
- **Cost of a node**: #violated diagonal constraints so far
- No heuristic

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



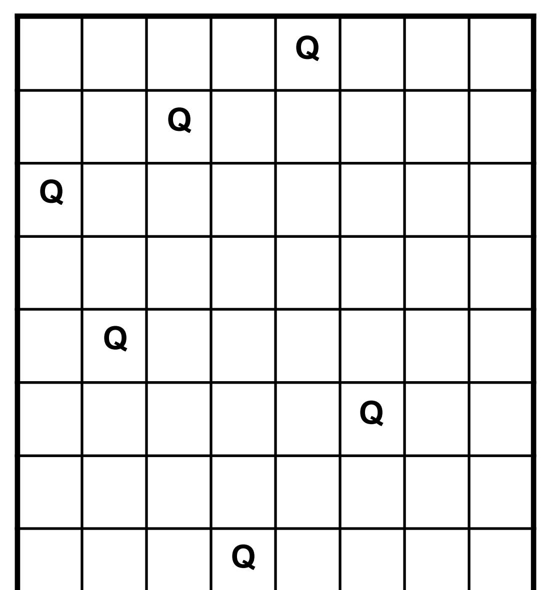
cost = 0

cost = 0



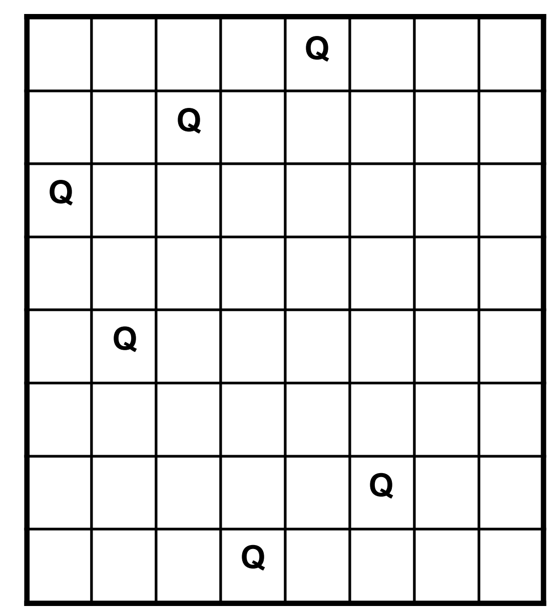
Depth first branch and bound will find a suboptimal solution here first (no way to tell at this point this is worse than right node)

cost = 1



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

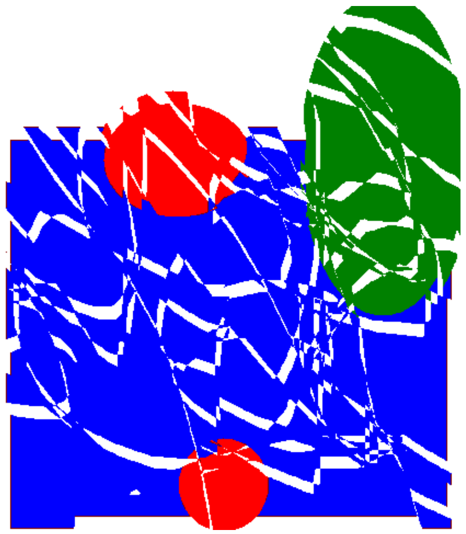
cost = 0

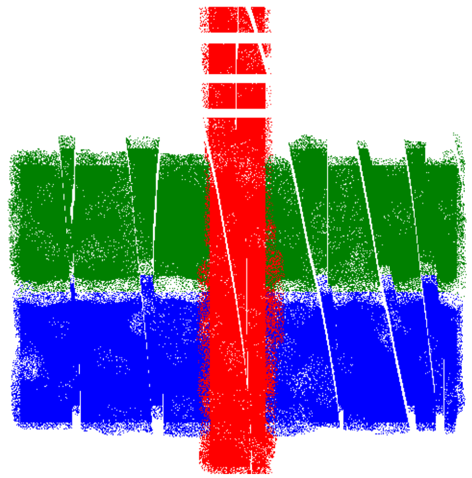


Optimal solution is down here (cost 0)

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

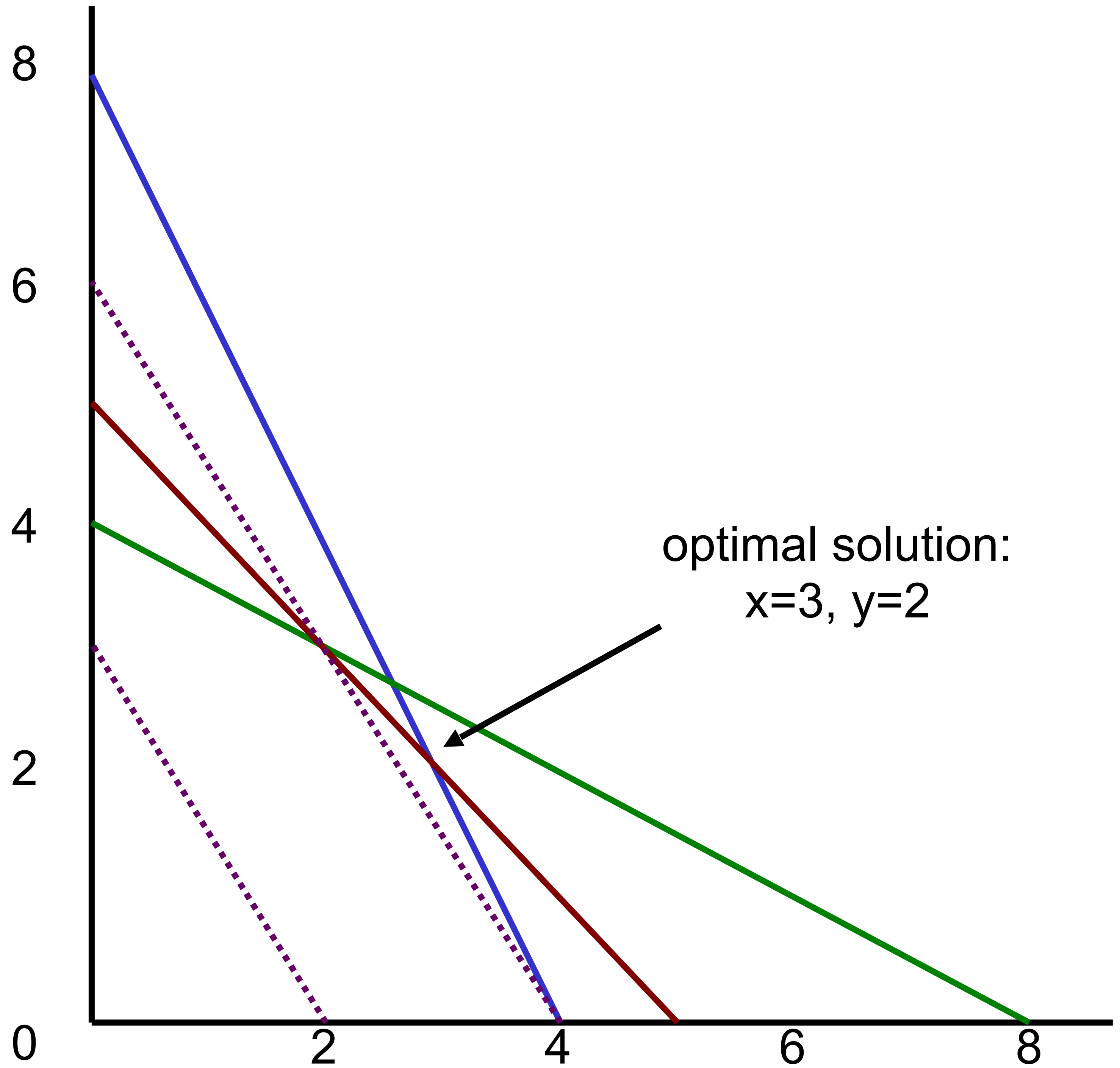
$4x + 2y \leq 16$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$

$y \geq 0$



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

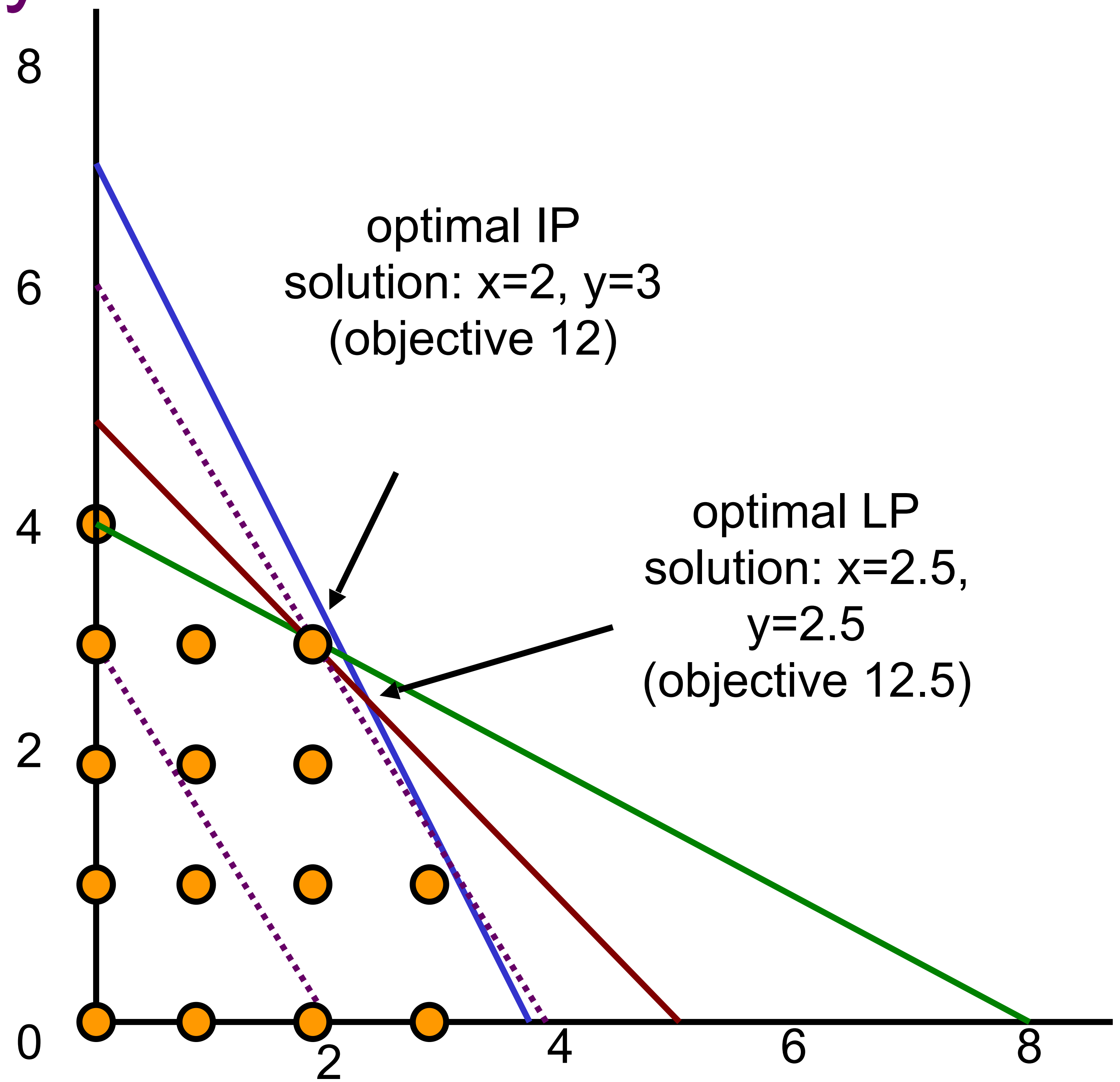
$4x + 2y \leq 15$

$x + 2y \leq 8$

$x + y \leq 5$

$x \geq 0$, integer

$y \geq 0$, integer



Mixed integer (linear) program

maximize 3x + 2y

subject to

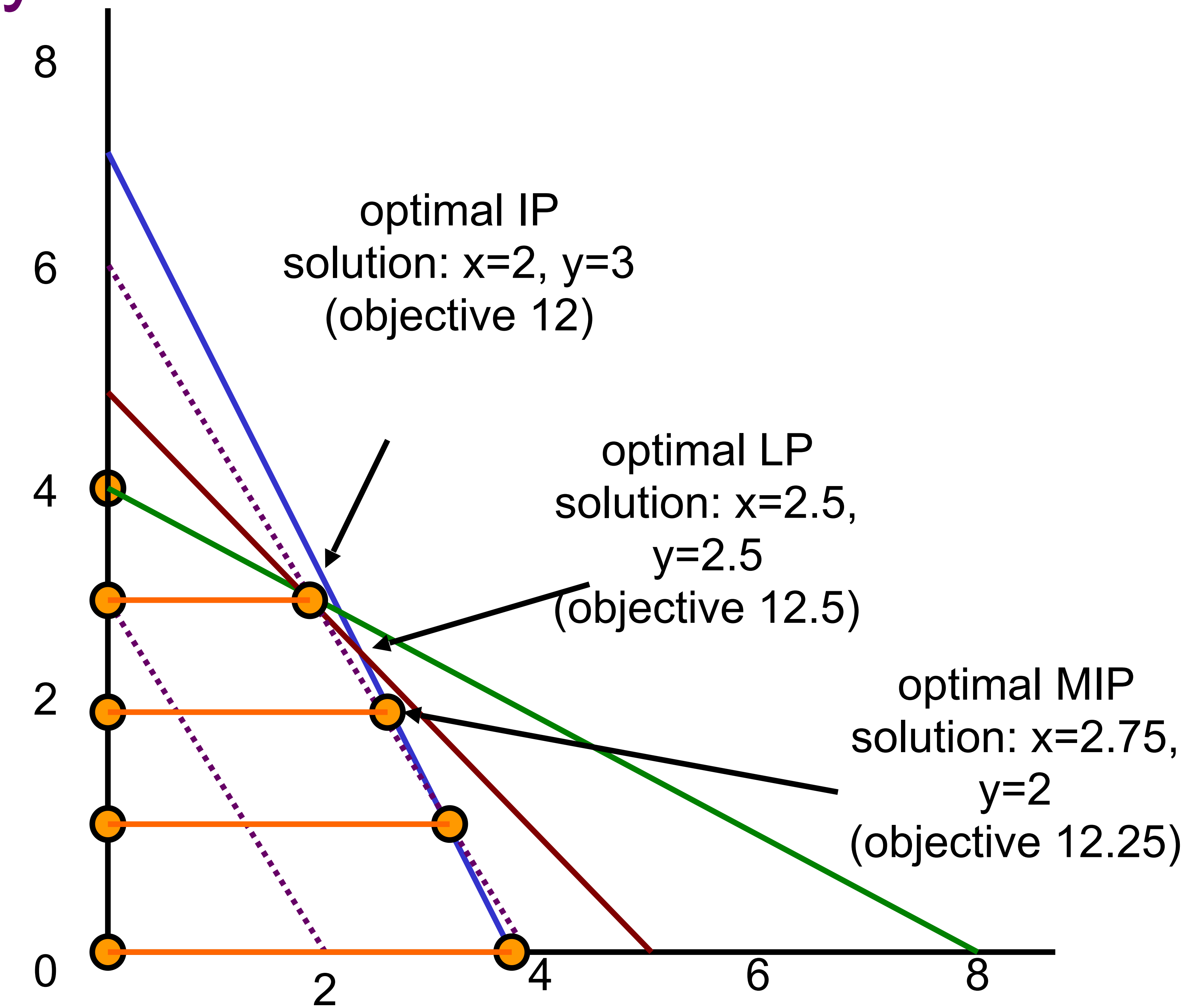
4x + 2y ≤ 15

x + 2y ≤ 8

x + y ≤ 5

x ≥ 0

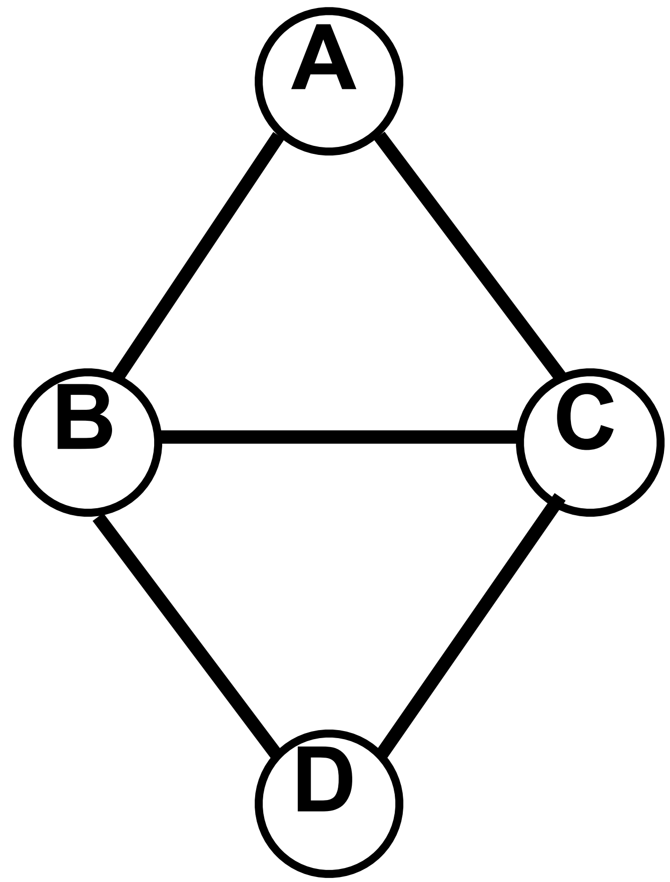
y ≥ 0, integer



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)

Graph coloring as an integer program



- Let's say $x_{B,green}$ is 1 if B is colored green, 0 otherwise
- Must have $0 \leq x_{B,green} \leq 1$, $x_{B,green}$ integer
  - shorthand: $x_{B,green}$ in {0,1}
- Constraint that B and C can't both be green: $x_{B,green} + x_{C,green} \leq 1$
- Etc.
- Solving integer programs is at least as hard as graph coloring, hence NP-hard (we have reduced graph coloring to IP)

Satisfiability as an integer program

(x_1 OR x_2 OR NOT(x_4)) AND (NOT(x_2) OR NOT(x_3)) 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 $x \geq k+1$

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, 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, obj = 12

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

LP solution: infeasible

maximize $3x + 2y$
subject to
$4x + 2y \leq 15$
$x + 2y \leq 8$
$x + y \leq 5$
$x \geq 3$
$y \leq 1$

LP solution: x=3.25, y=1, obj = 11.75

maximize $3x + 2y$
subject to
$4x + 2y \leq 15$
$x + 2y \leq 8$
$x + y \leq 5$
$x \geq 3$
$y \leq 1$
$x \geq 4$

LP solution: infeasible

maximize $3x + 2y$
subject to
$4x + 2y \leq 15$
$x + 2y \leq 8$
$x + y \leq 5$
$x \geq 3$
$y \leq 1$
$x \leq 3$

LP solution: x=3, y=1, obj = 11

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, obj = 12

if LP solution is integral, we are done

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, 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, obj = 12

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

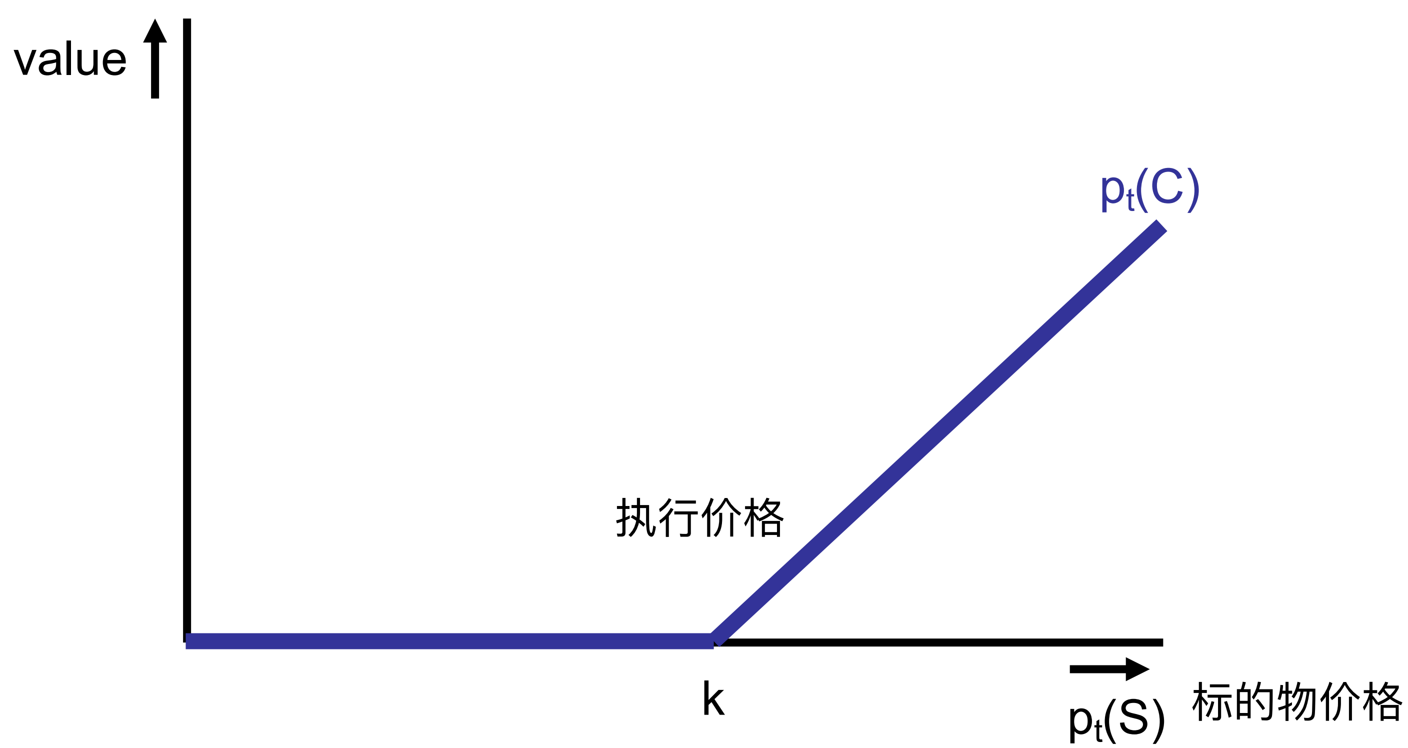
LP solution: x=2, y=3, obj = 12

done!

AI+金融市场

看涨期权（call option）

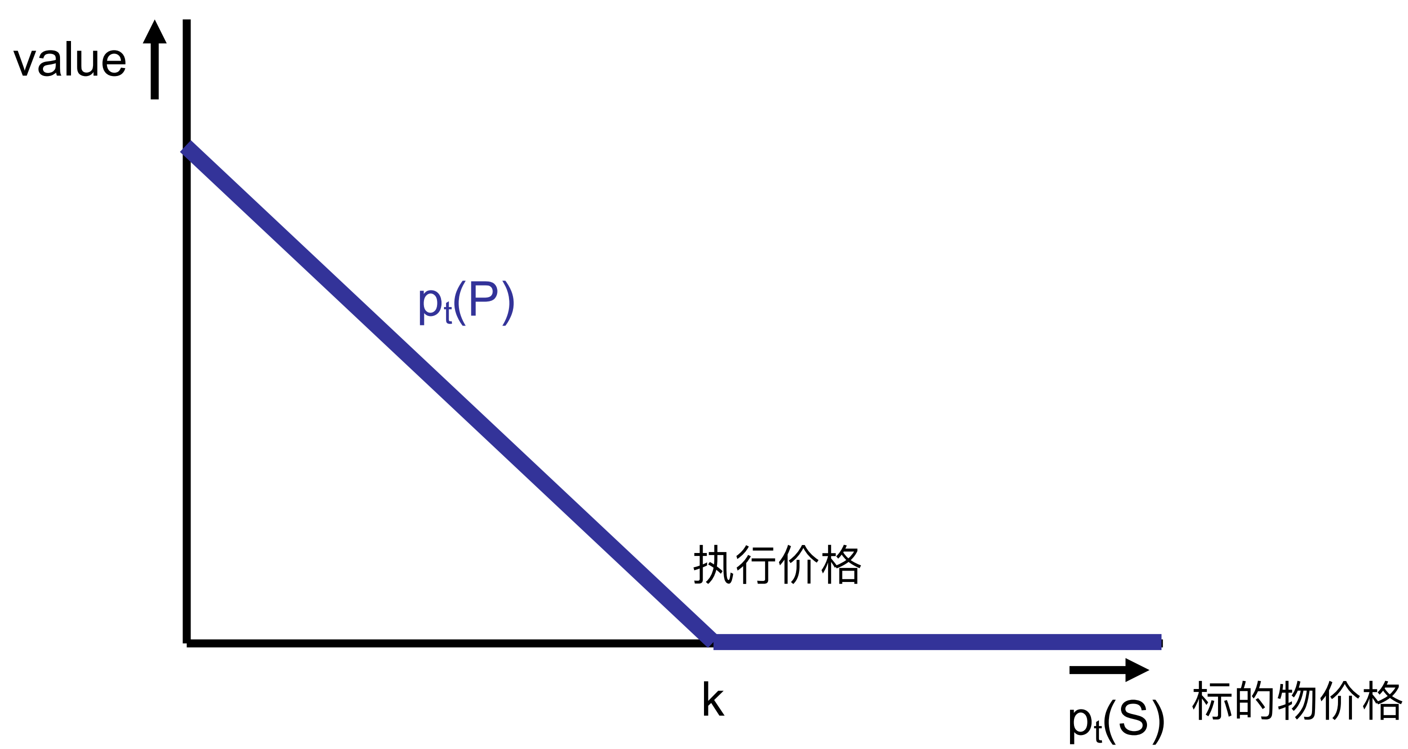
- A (European) call option C(S, k, t) gives you the right to buy stock S at (strike) price k on (expiry) date t
  - American call option can be exercised early
  - European one easier to analyze
- How much is a call option worth at time t (as a function of the price of the stock)?



看跌期权（put option）

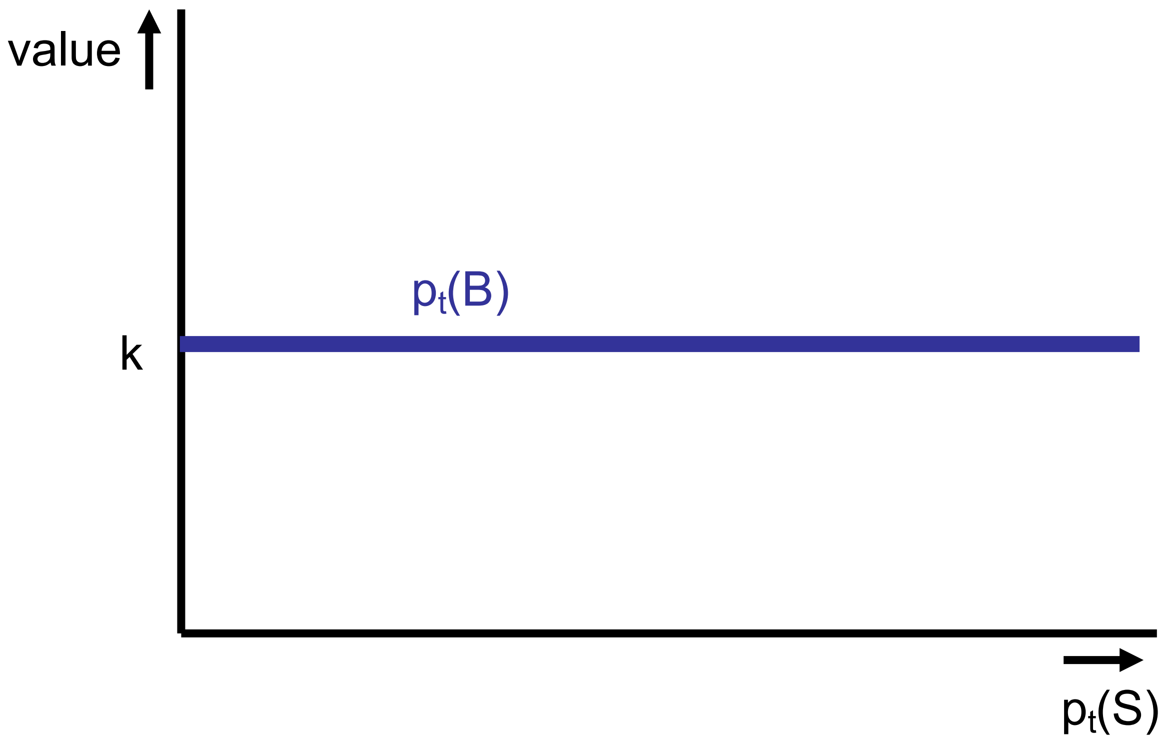
- A (European) put option P(S, k, t) gives you the right to sell stock S at (strike) price k on (expiry) date t

- How much is a put option worth at time t (as a function of the price of the stock)?

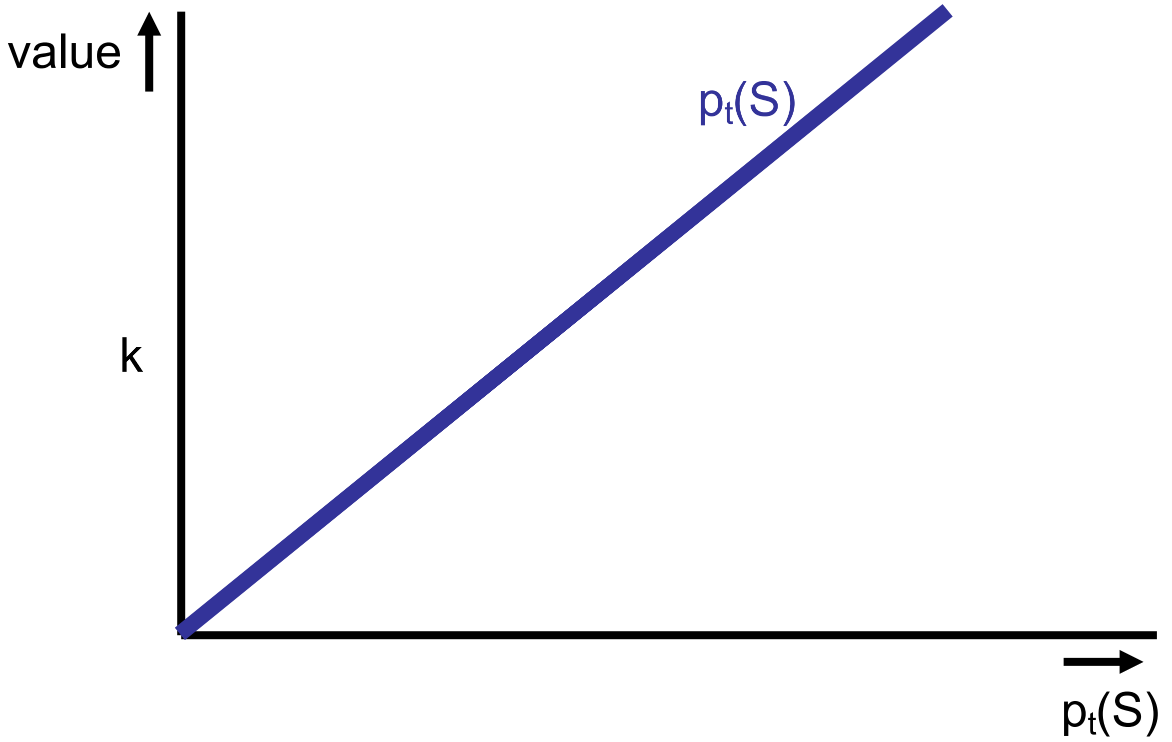


债券 bonds

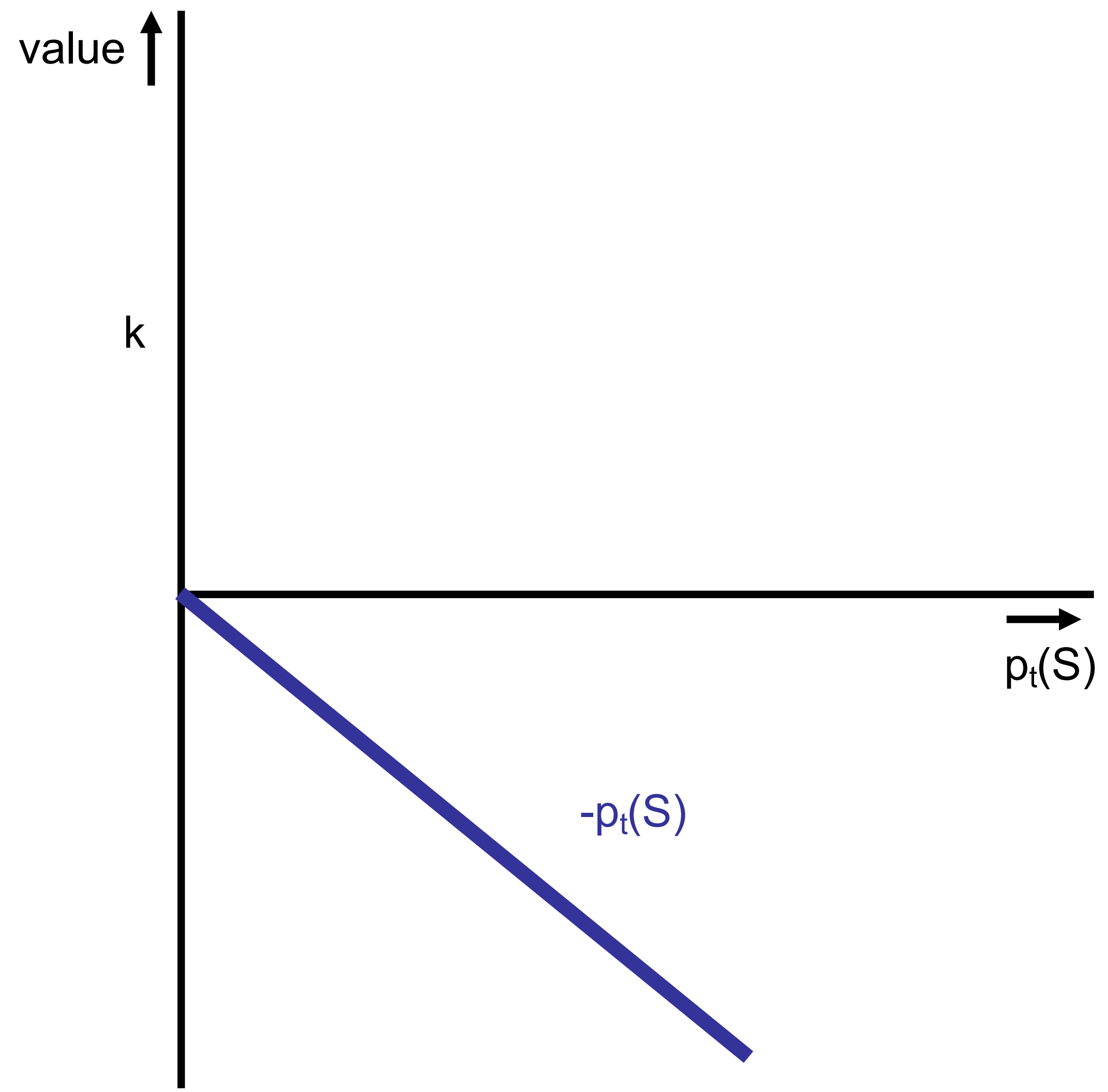
- A bond $B(k, t)$ pays off k at time t



股票 Stocks

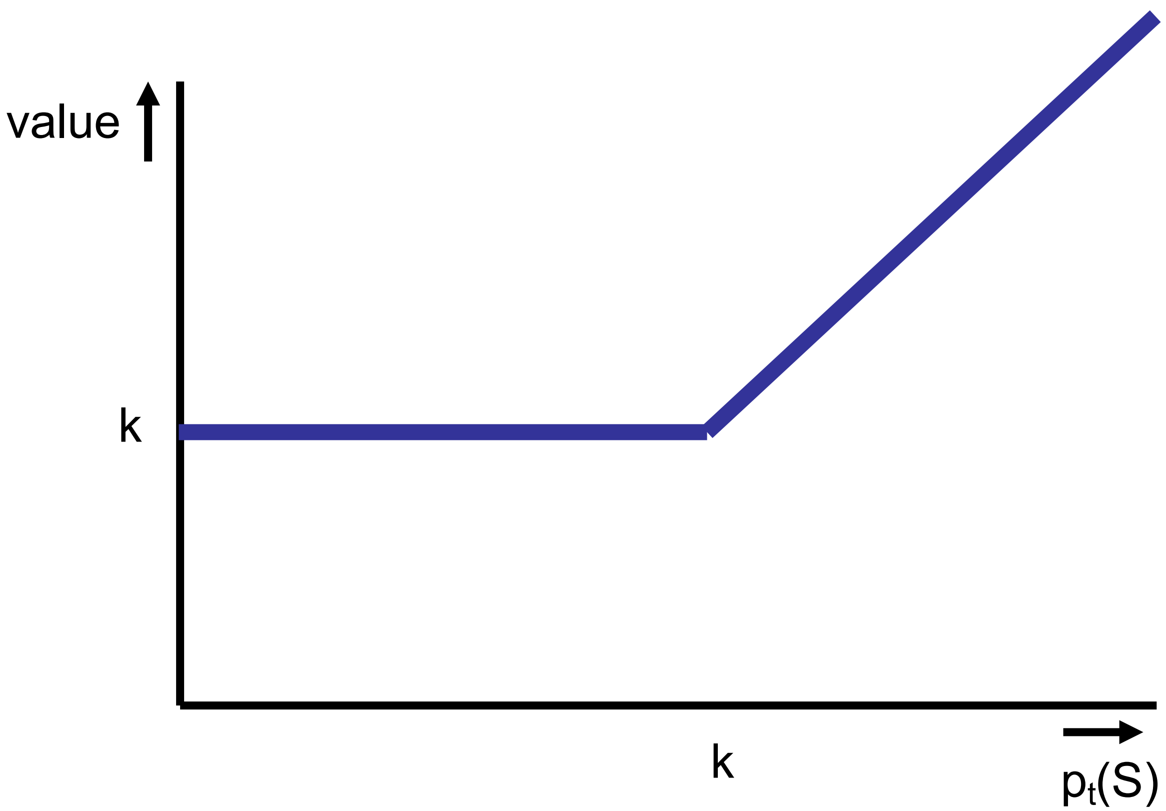


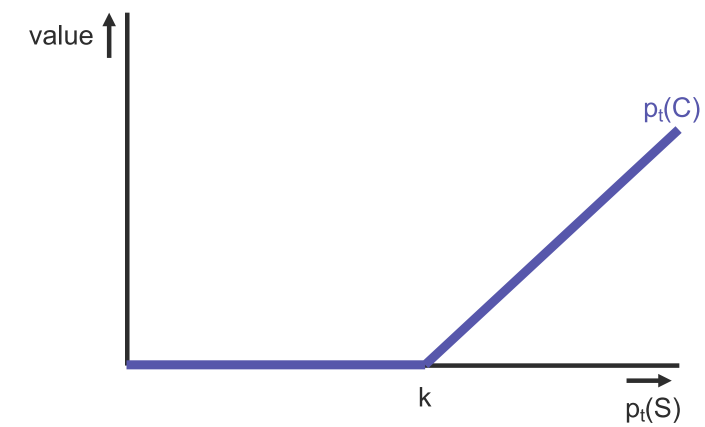
Selling a stock (short)



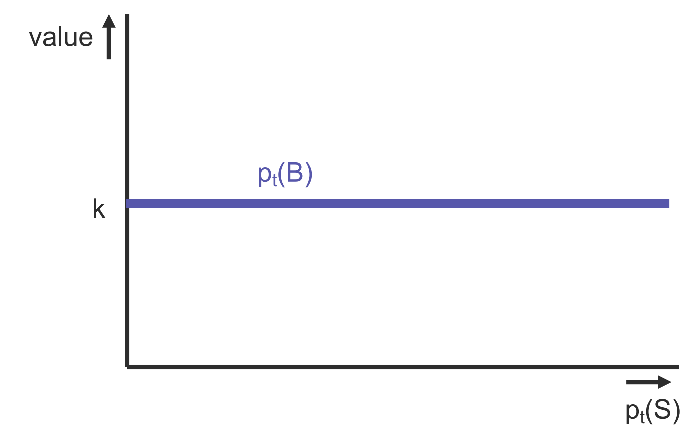
A portfolio

- One call option C(S, k, t) + one bond B(k, t)





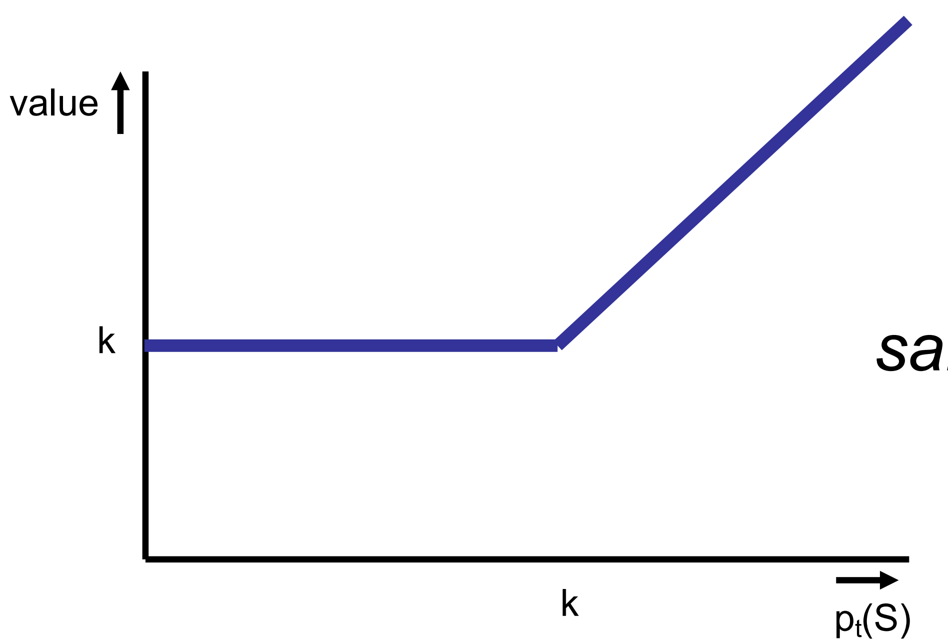
C(S, k, t)



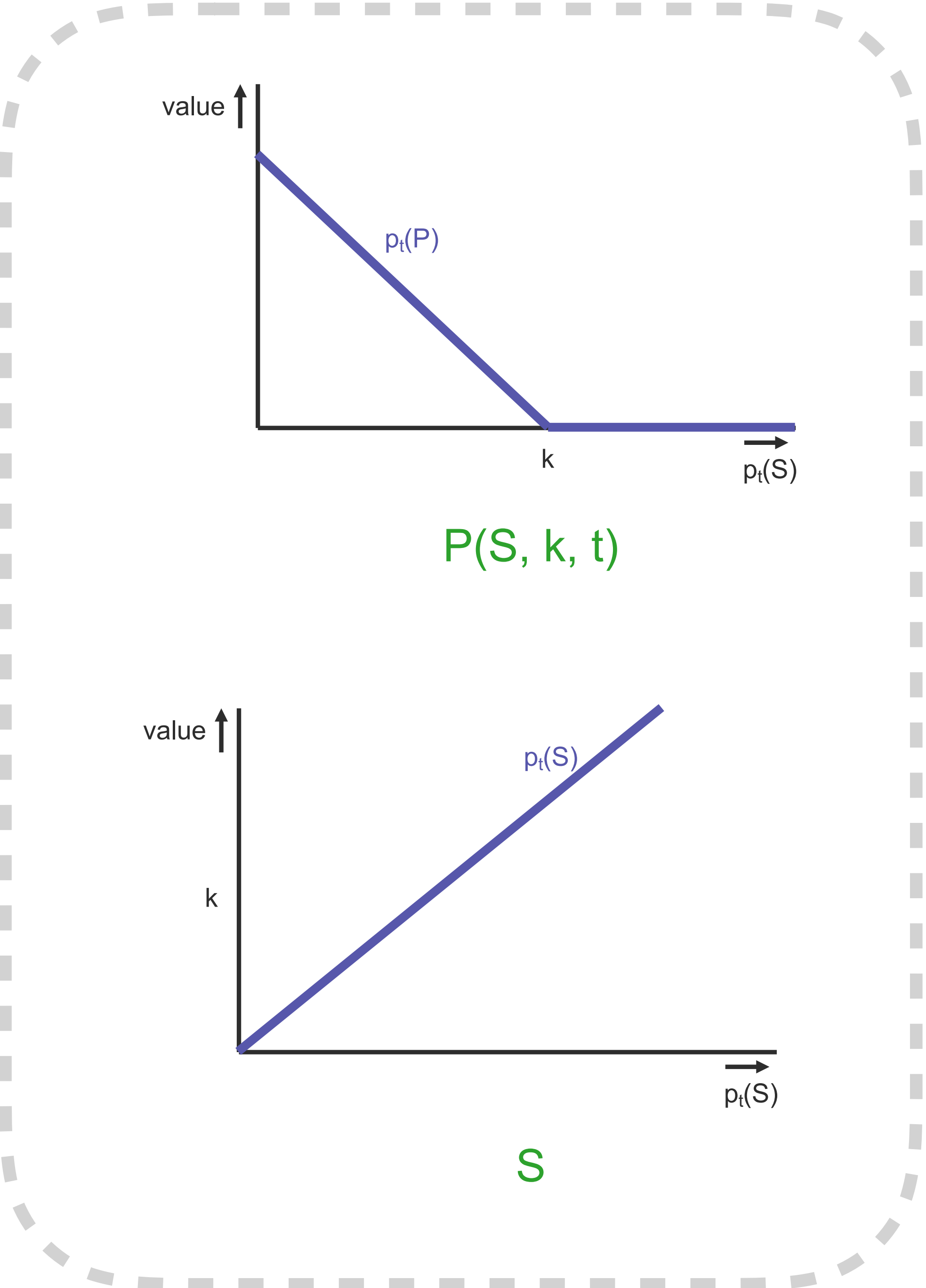
B(k, t)

Another portfolio

- One put option P(S, k, t) + one stock S



same thing!



Put-call parity（等价理论）

- C(S, k, t) + B(k, t) will have the same value at time t as P(S, k, t) + S (regardless of the value of S)
- Assume stocks pay no dividends
- Then, portfolio should have the same value at any time before t as well
- I.e., for any t' < t, it should be that $p_{t'}(C(S, k, t)) + p_{t'}(B(k, t)) = p_{t'}(P(S, k, t)) + p_{t'}(S)$

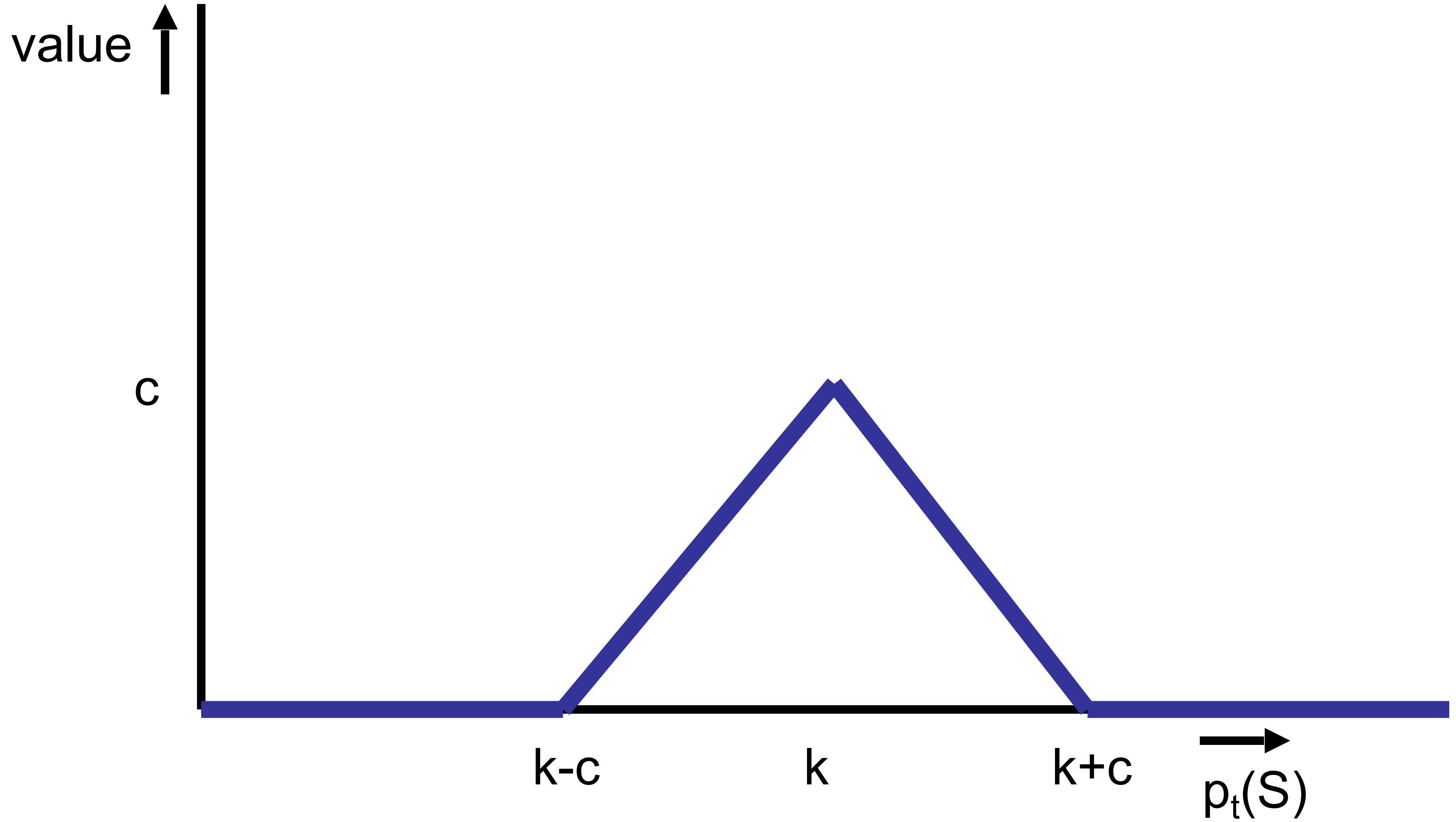
- Arbitrage argument: suppose (say) $p_{t'}(C(S, k, t)) + p_{t'}(B(k, t)) < p_{t'}(P(S, k, t)) + p_{t'}(S)$
- Then: buy C(S, k, t) + B(k, t), sell (short) P(S, k, t) + S
- Value of portfolio at time t is 0
- Guaranteed profit!

Another perspective: auctioneer

- Auctioneer receives buy and sell offers, has to choose which to accept
- E.g.: offers received: buy(S, \$10); sell(S, \$9)
- Auctioneer can accept both offers, profit of \$1
- E.g. (put-call parity):
  - sell(C(S, k, t), \$3)
  - sell(B(k, t), \$4)
  - buy(P(S, k, t), \$5)
  - buy(S, \$4)
- Can accept all offers at no risk!

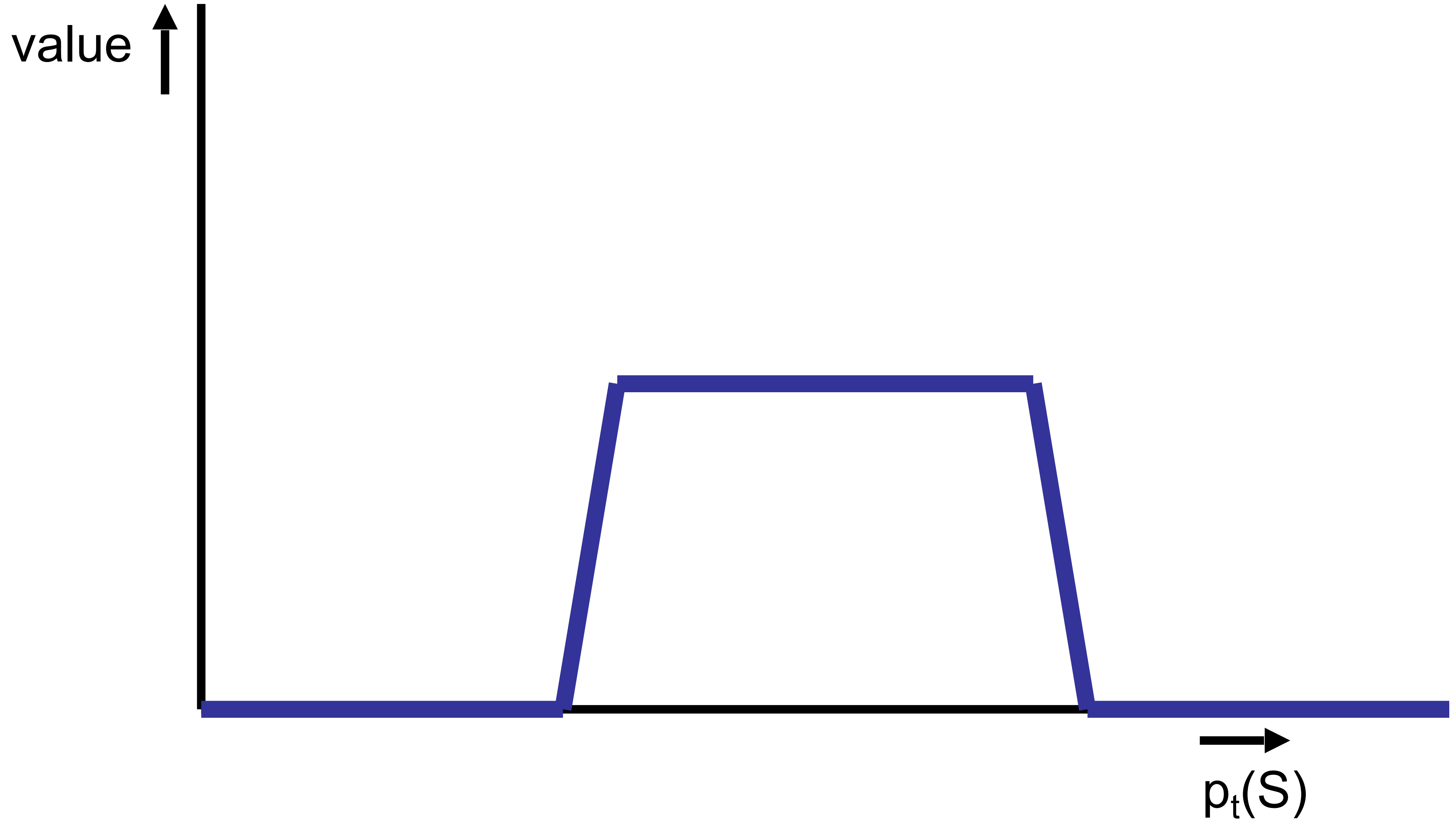
"Butterfly" portfolio

- 1 call at strike price k-c
- -2 calls at strike k
- 1 call at strike k+c



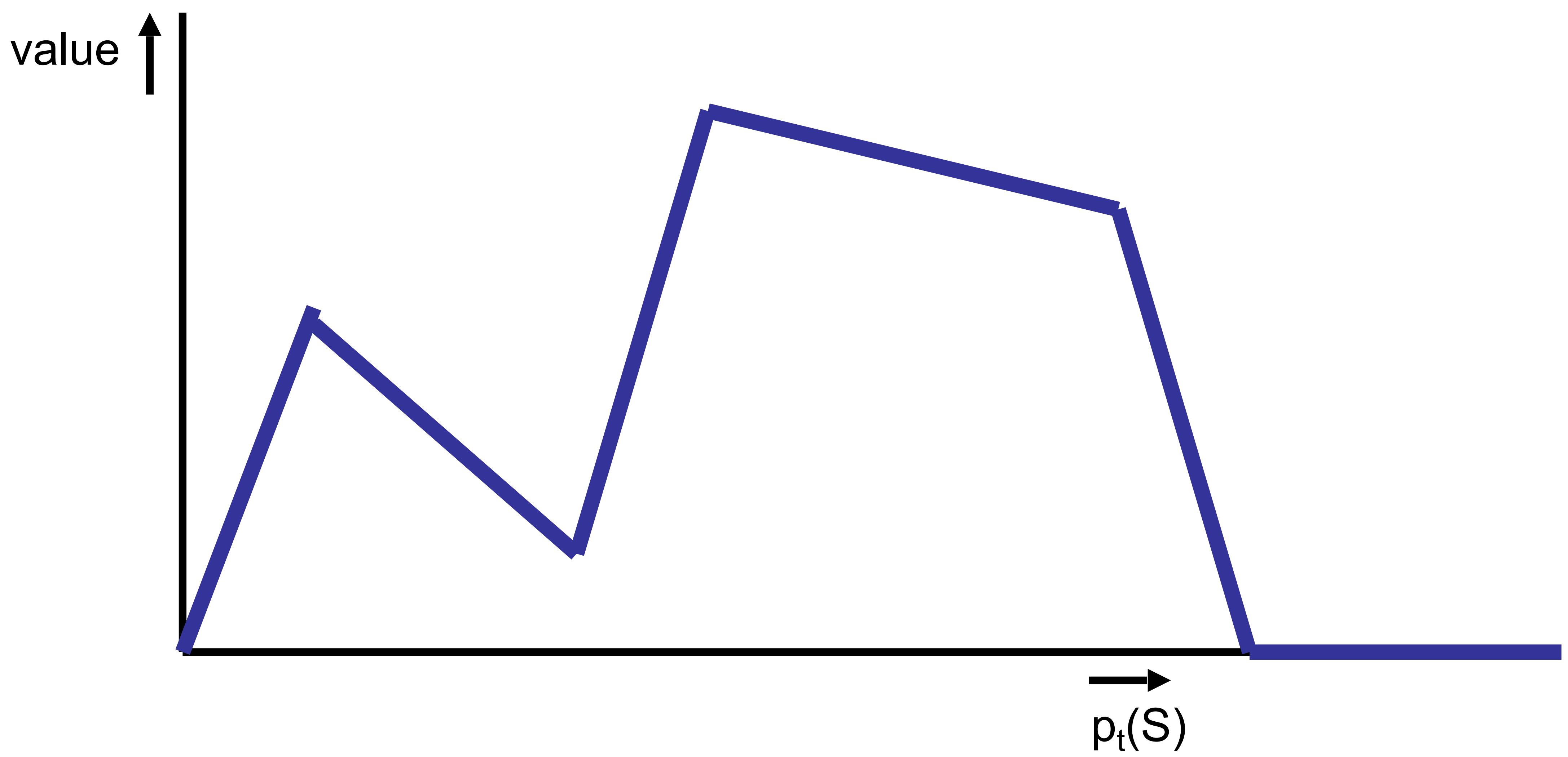
Another portfolio

- Can we create this portfolio?



Yet another portfolio

- How about this one?



Securities conditioned on finite set of outcomes

- E.g., InTrade: security that pays off 1 if Trump is the Republican nominee in 2016
- Can we construct a portfolio that pays off 1 if Clinton is the Democratic nominee AND Trump is the Republican nominee?

	Trump not nom.	Trump nom.
Clinton not nom.	\$0	\$0
Clinton nom.	\$0	\$1

Arrow-Debreu securities

- Suppose S is the set of all states that the world can be in tomorrow
- For each s in S, there is a corresponding Arrow-Debreu security that pays off 1 if s happens, 0 otherwise
- E.g., s could be: Clinton is nominee and Trump is nominee and S_1 is at \$4 and S_2 at \$5 and butterfly 432123 flaps its wings in Peru and…
- Not practical, but conceptually useful
- Can think about Arrow-Debreu securities within a domain (e.g., states only involve stock trading prices)
- Practical for small number of states

With Arrow-Debreu securities you can do anything…

- Suppose you want to receive \$6 in state 1, \$8 in state 2, \$25 in state 3

- … simply buy 6 AD securities for state 1, 8 for state 2, 25 for state 3

- Linear algebra: Arrow-Debreu securities are a basis for the space of all possible securities

The auctioneer problem

- Tomorrow there must be one of 
- Agent 1 offers \$5 for a security that pays off \$10 if  
- Agent 2 offers \$8 for a security that pays off \$10 if  
- Agent 3 offers \$6 for a security that pays off \$10 if 
- Can we accept some of these at offers at no risk?

Reducing auctioneer problem to ~combinatorial exchange winner determination problem

- Let (x, y, z) denote payout under    respectively
- Previous problem's bids:
  - 5 for (0, 10, 10)
  - 8 for (10, 0, 10)
  - 6 for (10, 0, 0)
- Equivalently:
  - (-5, 5, 5)
  - (2, -8, 2)
  - (4, -6, -6)
- Sum of accepted bids should be (≤0, ≤0, ≤0) to have no risk
- Sometimes possible to partially accept bids

A bigger instance (4 states)

- Objective: maximize our worst-case profit
- 3 for (0, 0, 11, 0)
- 4 for (0, 2, 0, 8)
- 5 for (9, 9, 0, 0)
- 3 for (6, 0, 0, 6)
- 1 for (0, 0, 0, 10)

- What if they are partially acceptable?

Settings with large state spaces

- Large = exponentially large
  - Too many to write down
- Examples:
- $S = S_1 \times S_2 \times \ldots S_n$
  - E.g., S_1 = {Clinton not nom., Clinton nom.}, S_2 = {Trump not nom., Trump nom.}, S = {(-C, -T), (-C, +T), (+C, -T), (+C, +T)}
  - If all S_i have the same size k, there are k^n different states
- S is the set of all rankings of n candidates
  - E.g., outcomes of a horse race
  - n! different states (assuming no ties)

Bidding languages

- How should trader (bidder) express preferences?
- Logical bidding languages [Fortnow et al. 2004]:
  - (1) "If Trump nominated OR (Cruz nominated AND Clinton nominated), I want to receive \$10; I'm willing to pay \$6 for this."
- If the state is a ranking [Chen et al. 2007] :
  - (2a) "If horse A ranks 2nd, 3rd, or 4th I want to receive \$10; I'm willing to pay \$6 for this."
  - (2b) "If one of horses A, C, D rank 2nd, I want to receive \$10; I'm willing to pay \$6 for this."
  - (2c) "If horse A ranks ahead of horse C, I want to receive \$10; I'm willing to pay \$6 for this."
- Winner determination problem is NP-hard for all of these, except for (2a) and (2b) which are in P if bids can be partially accepted

谢谢！