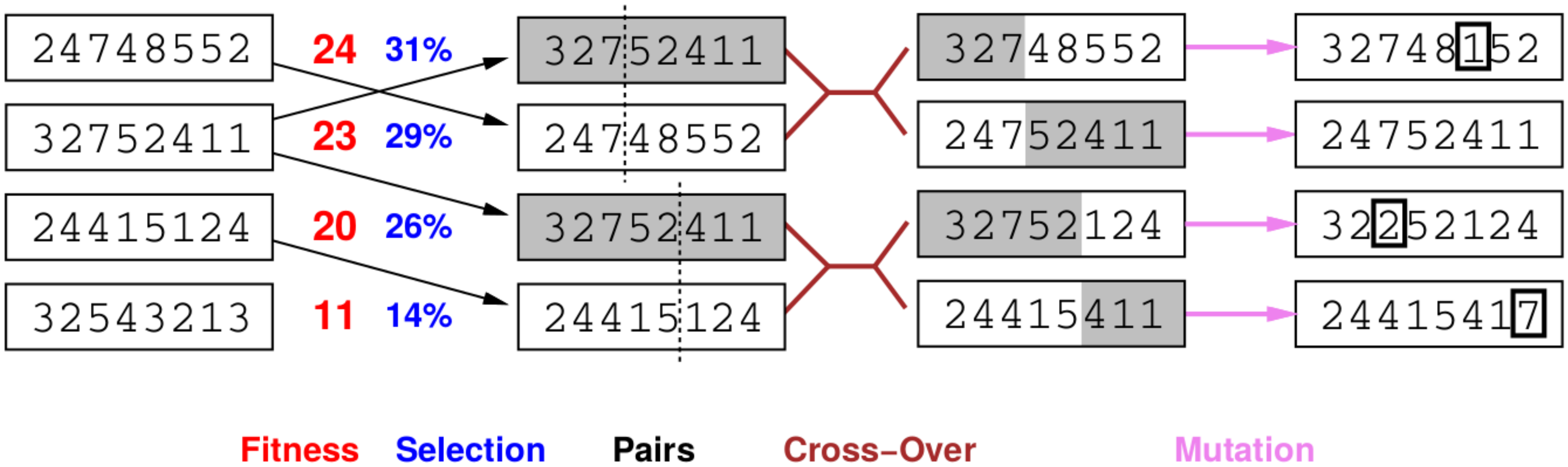
Genetic algorithms

= stochastic local beam search + generate successors from **pairs** of states



Game playing

in competitive multi-agent problems

Reminder: Taxi example. How do you define multi-agent problems?

Outline

◇ Games

◇ Perfect play
  – minimax decisions
  – α–β pruning

◇ Resource limits and approximate evaluation

◇ Games of chance

◇ Games of imperfect information

From the textbook: "Game playing was one of the first tasks undertaken by AI. ... to the point that machines have surpassed humans in ... The main exception is Go, in which computers perform at the amateur level".
search tree of chess: 35^100 in average.

Games vs. search problems

"Unpredictable" opponent $\Rightarrow$ solution is a strategy specifying a move for every possible opponent reply

Time limits $\Rightarrow$ unlikely to find goal, must approximate

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Require the ability to make some decision even when calculating the optimal decision is infeasible.

Types of games

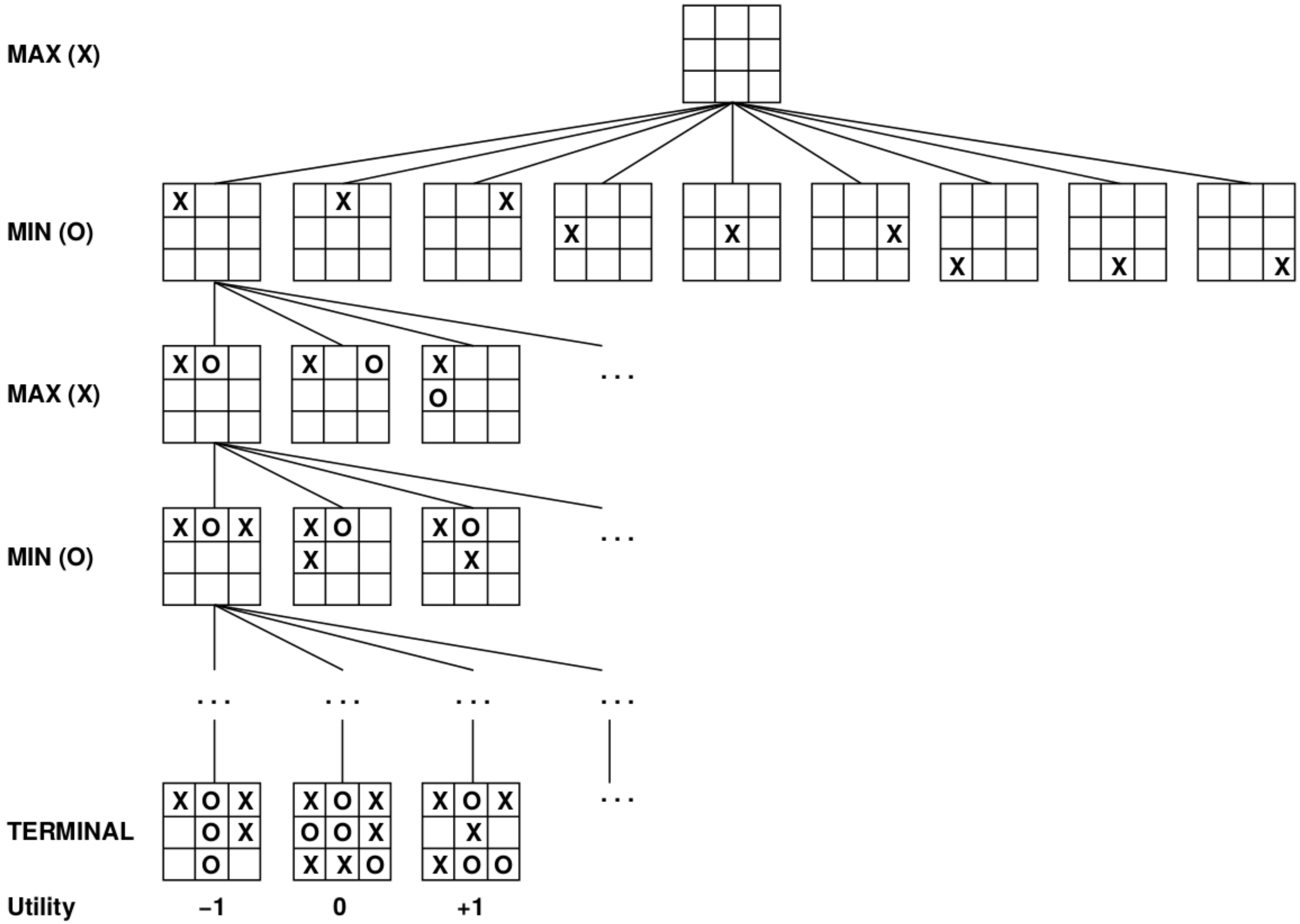
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Define game formally:
- initial state
- successor function
- terminal test
- utility function

"zero-sum" games are considered here.

2-players: MAX and MIN. Max moves first

Game tree (2-player, deterministic, turns)



minimax value:High values are assumed to be good for MAX (&bad for MIN).
MAX should find a "contingent" strategy.

Optimal strategy: leads to outcomes at least as good as any other strategy when one is playing a perfect opponent.

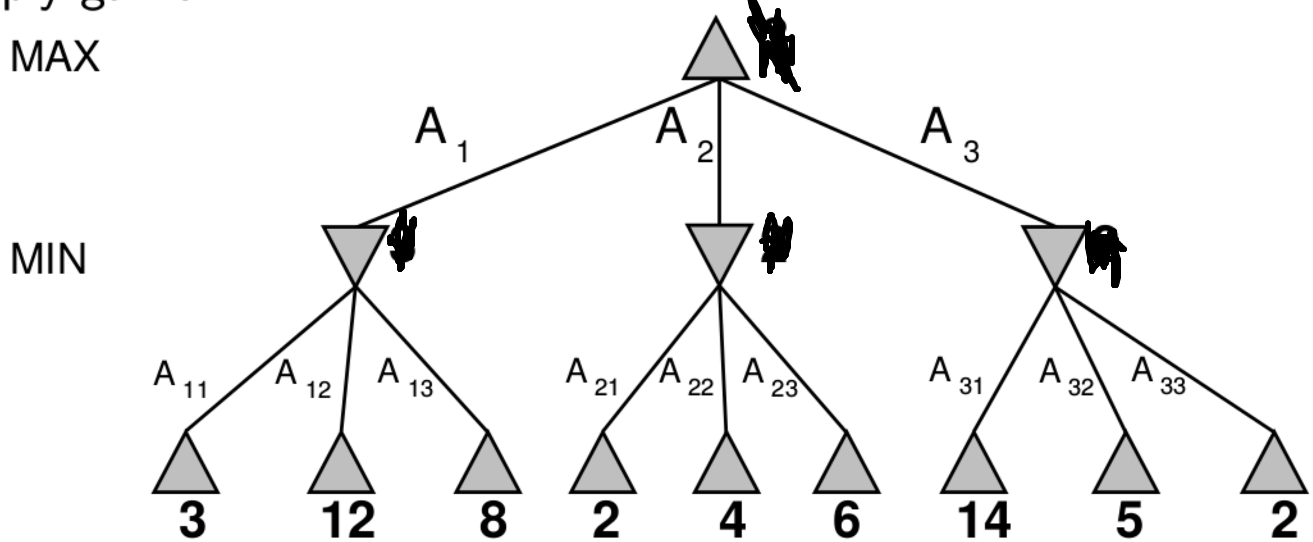
Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
= best achievable payoff against best play

assuming both players play optimally.

E.g., 2-ply game:



MAX prefers to a state with max value, MIN prefers min

MINIMAX-VALUE (n) = Utility (n) if n is terminal
max MINIMAX-VAL (s) where s is successor and n is MAX node
min MINIMAX-VAL (s) where s is successor and n is MIN node

How about minimax decision at the root?

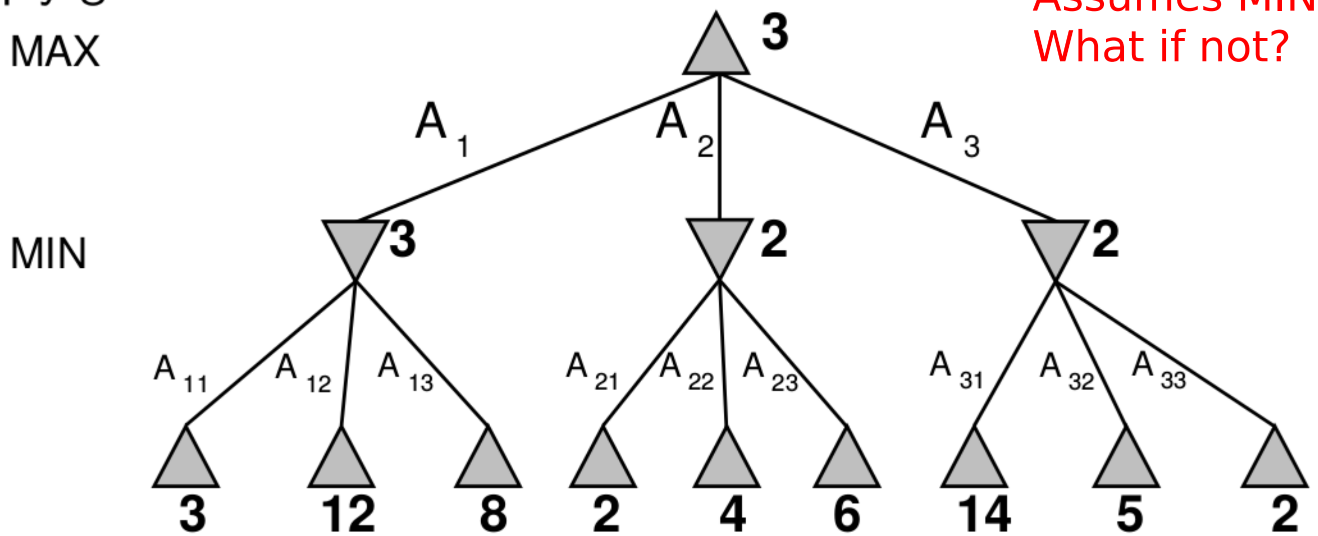
Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
= best achievable payoff against best play

E.g., 2-ply game:

Assumes MIN plays optimal.
What if not?



How about minimax decision at the root?

Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for a, s in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN(v, MAX-VALUE(s))
  return v

Properties of minimax

Complete??

Properties of minimax

Complete?? Only if tree is finite

Optimal??

Properties of minimax

Complete?? Yes, if tree is finite

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? .

What type of exploration?

Properties of minimax

Complete?? Yes, if tree is finite

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games

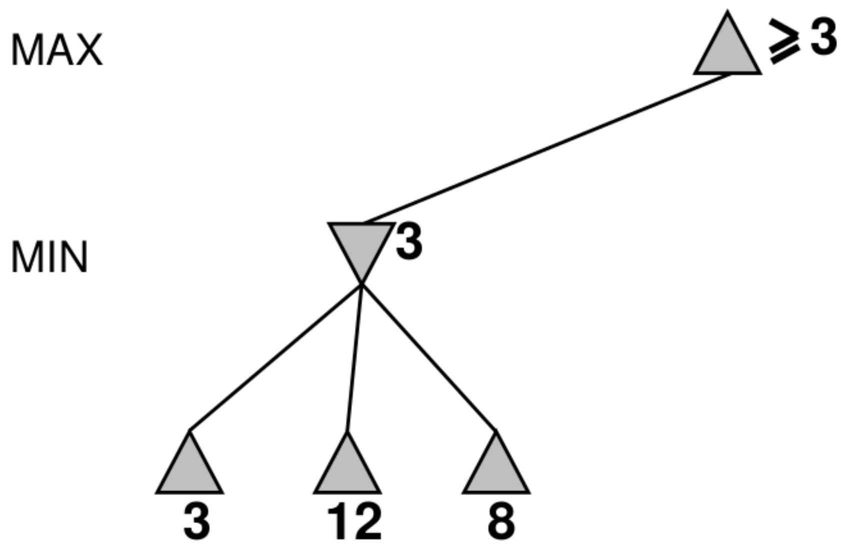
$\Rightarrow$ exact solution completely infeasible

2 questions:

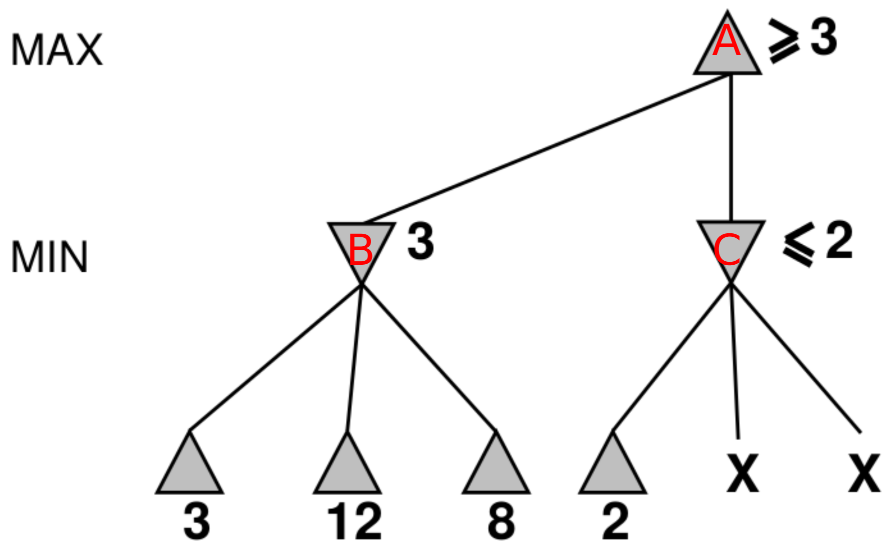
But do we need to explore every path?

How about multi-player games?

α–β pruning example

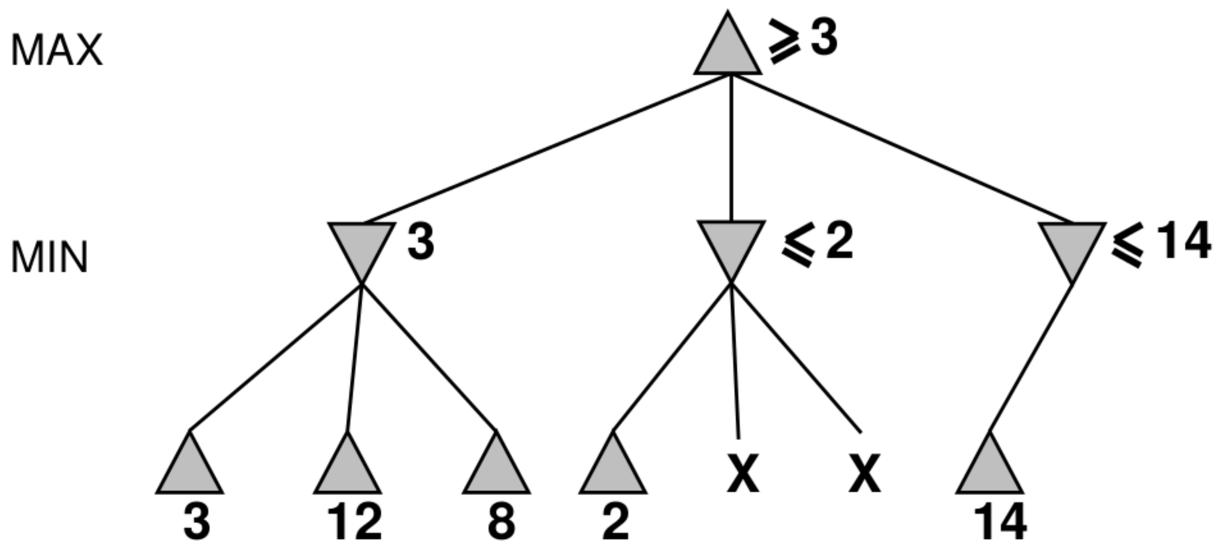


α–β pruning example

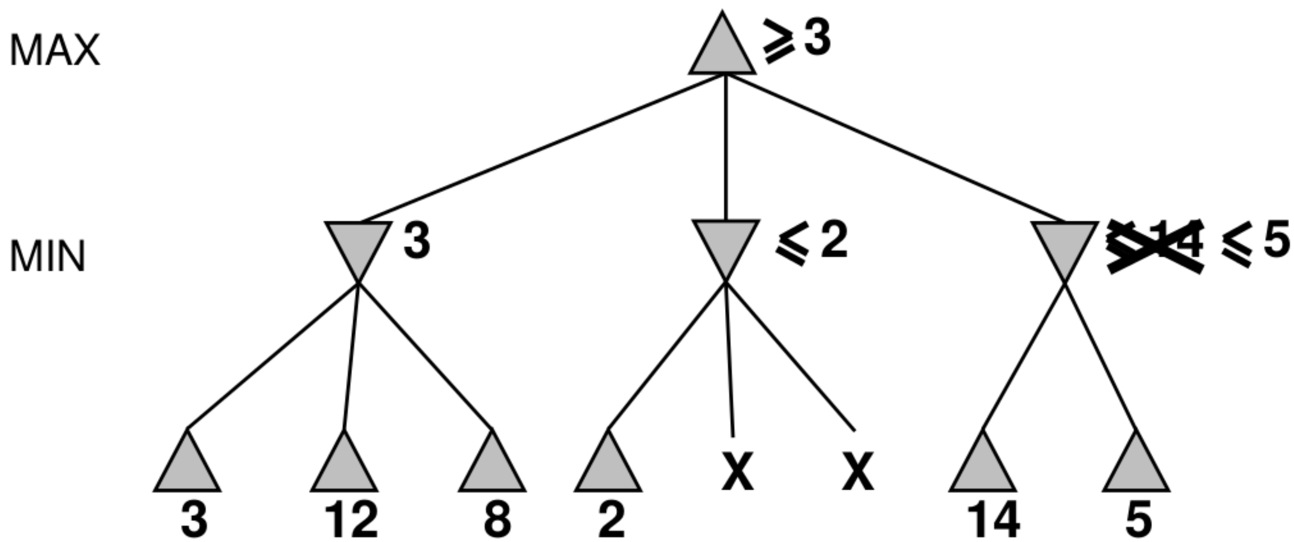


A would never choose C.

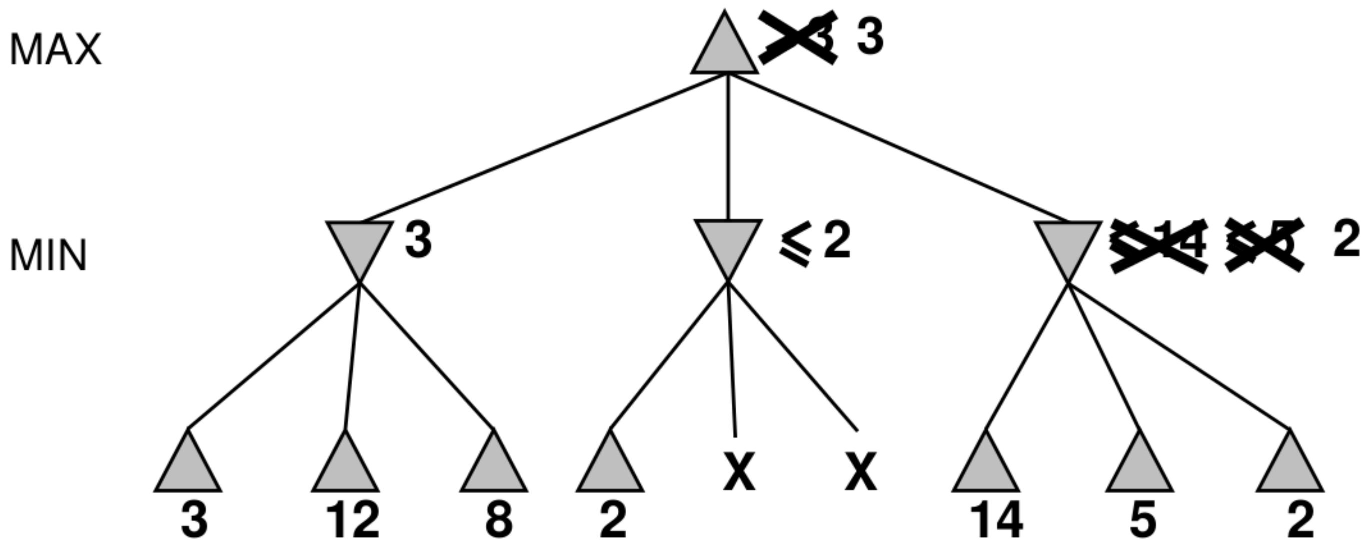
α–β pruning example



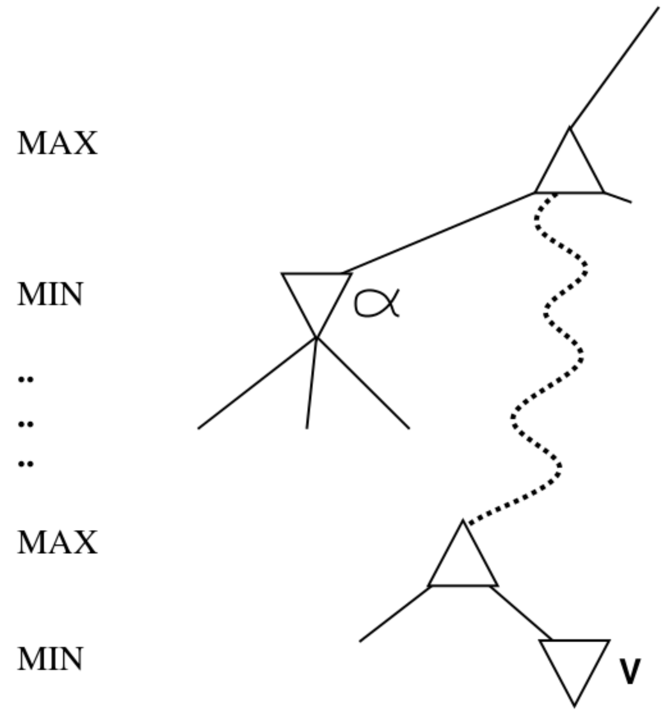
α–β pruning example



α–β pruning example



Why is it called α–β?



α is the best value (to MAX) found so far off the current path

If V is worse than α, MAX will avoid it $\Rightarrow$ prune that branch
(i.e. when we learn enough aboun v)

Define β similarly for MIN

The α–β algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

alpha: the value of the best (highest) choice found at any choice point along the path for MAX
beta: the value of the best (lowest) choice found at any choice point along the path for MIN

Properties of α–β

Pruning **does not** affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$
$\Rightarrow$ **doubles** solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Unfortunately, 35^{50} is still impossible!

Resource limits

Standard approach:

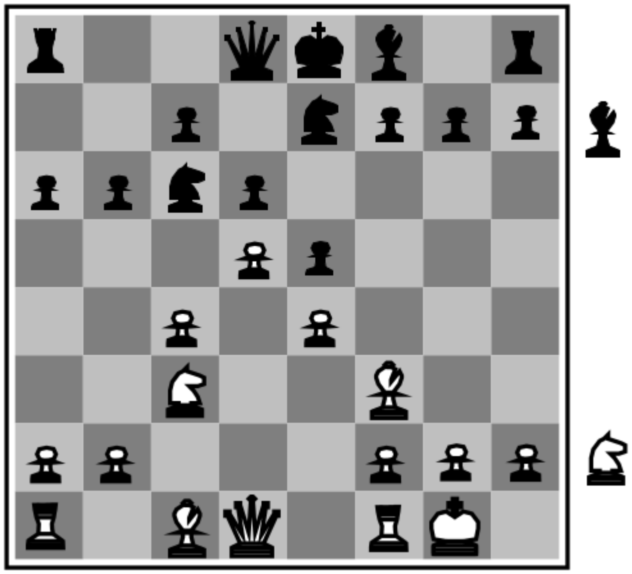
- Use CUTOFF-TEST instead of TERMINAL-TEST
  e.g., depth limit (perhaps add quiescence search)
- Use EVAL instead of UTILITY
  i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore 10^4 nodes/second
$\Rightarrow$ 10^6 nodes per move $\approx 35^{8/2}$
$\Rightarrow$ α–β reaches depth 8 $\Rightarrow$ pretty good chess program

1- EVAL should order the terminal nodes the same way as utility
2- Computation time should be short
3- Should be strongly correlated with the actual chances of winning

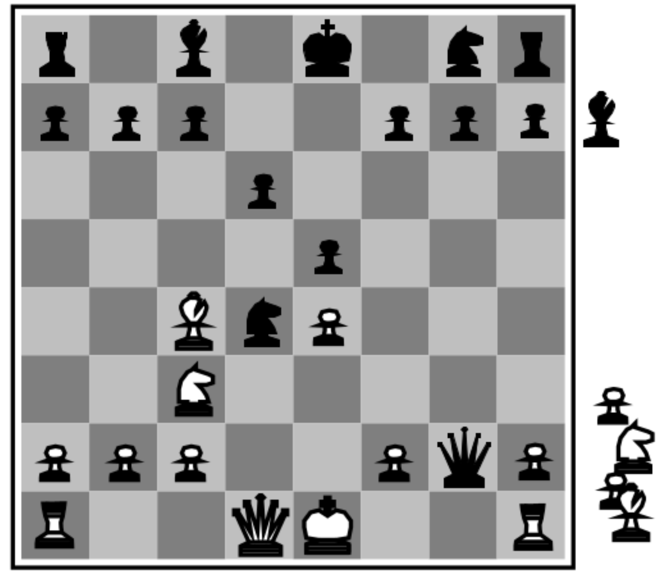
EVAL (x) = .. for the chess?
- find some categories, find expected value of winning for each category
.... too many categories, a lot of experience.
- what else?

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically linear weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

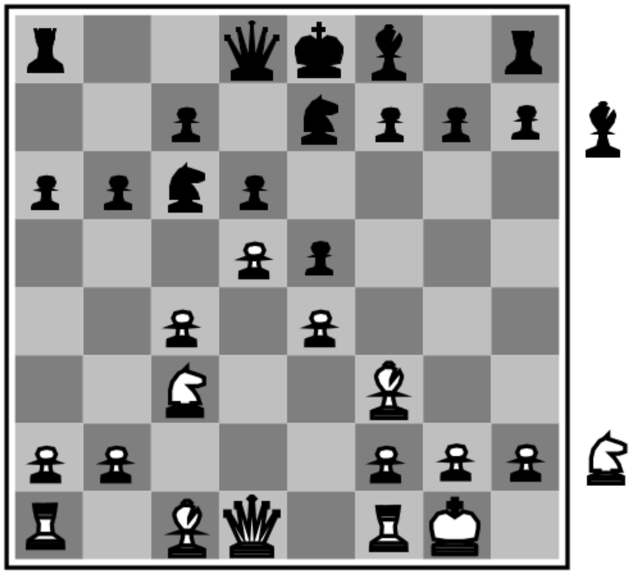
pawn: 1
knight or bishop: 3
rook: 5

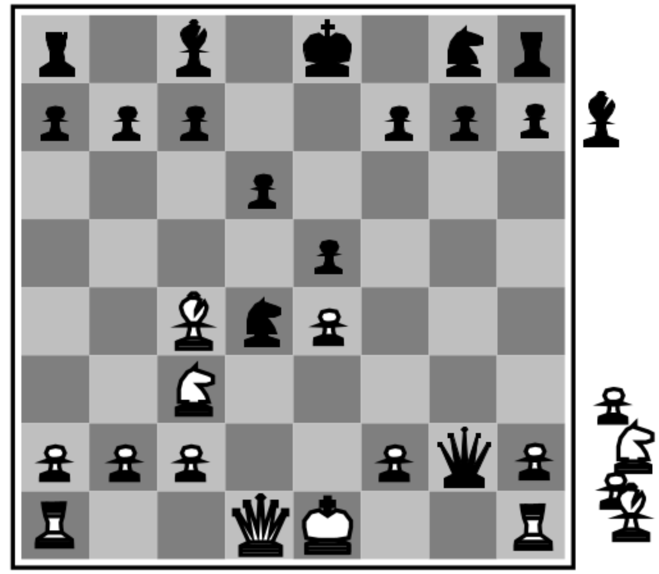
Try to include the following information: a pair of bishops worth more than twice the value of the bishop
No rule information is included in the evaluation..

Where to cut-off?
fixed depth? more robust: iterative deepening until time limit?
apply only in quiescent positions: unlikely exhibit significant changes in the value.

--> quiescence search

Evaluation functions





Black to move

White slightly better

White to move

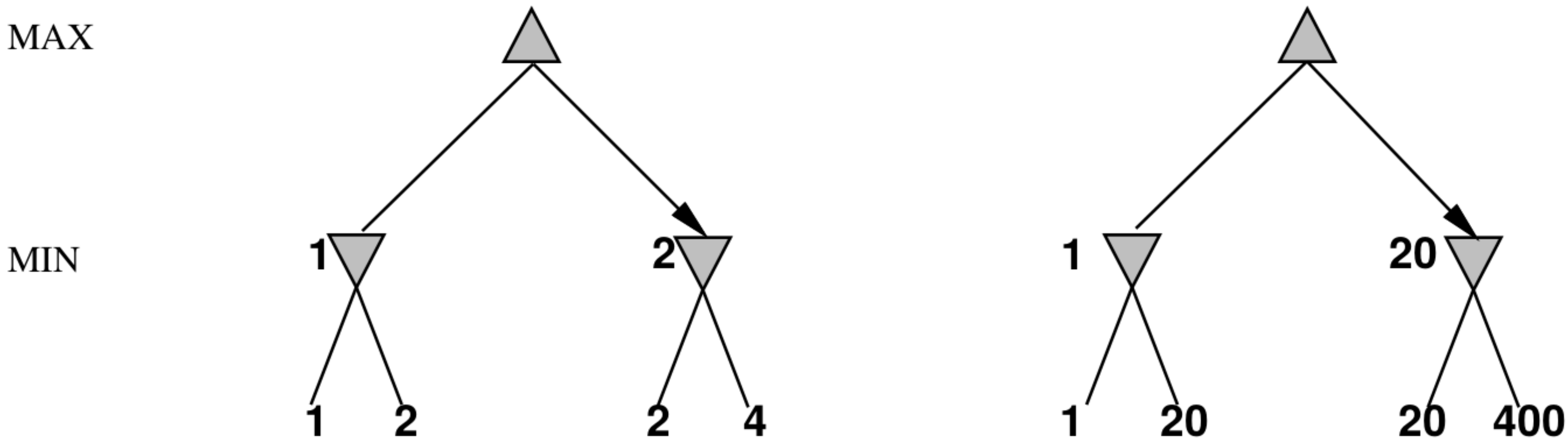
Black winning

For chess, typically linear weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Digression: Exact values don't matter



Behaviour is preserved under any **monotonic** transformation of EVAL

Only the order matters:

payoff in deterministic games acts as an ordinal utility function

Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

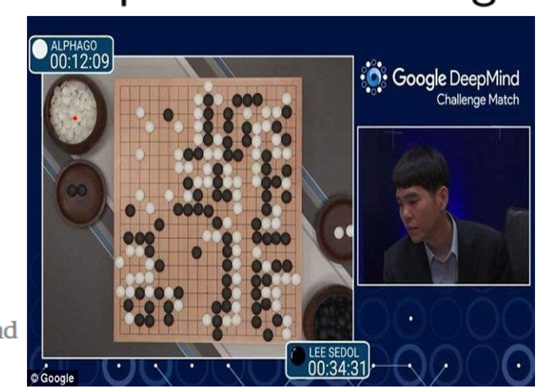
Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

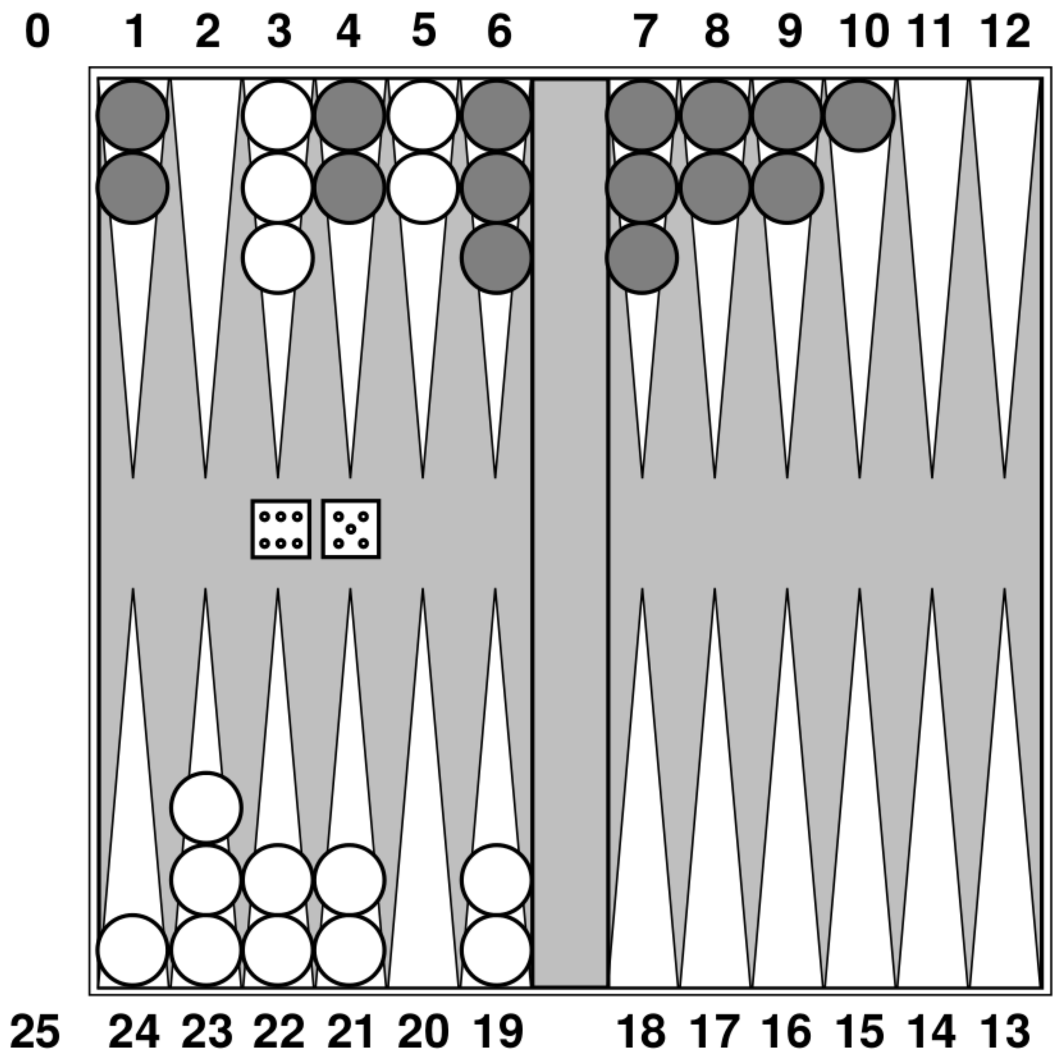
Not anymore..

AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

DeepMind's artificial intelligence astonishes fans to defeat human opponent and offers evidence computer software has mastered a major challenge

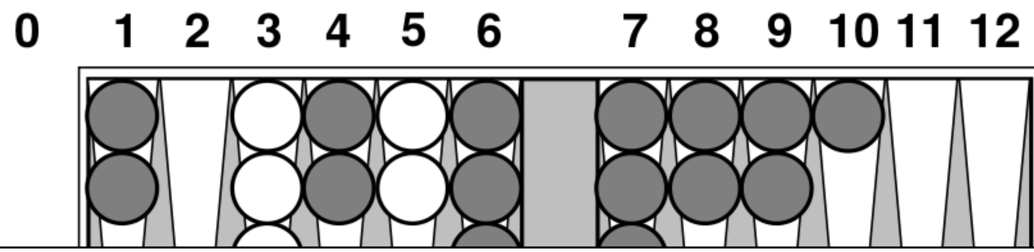


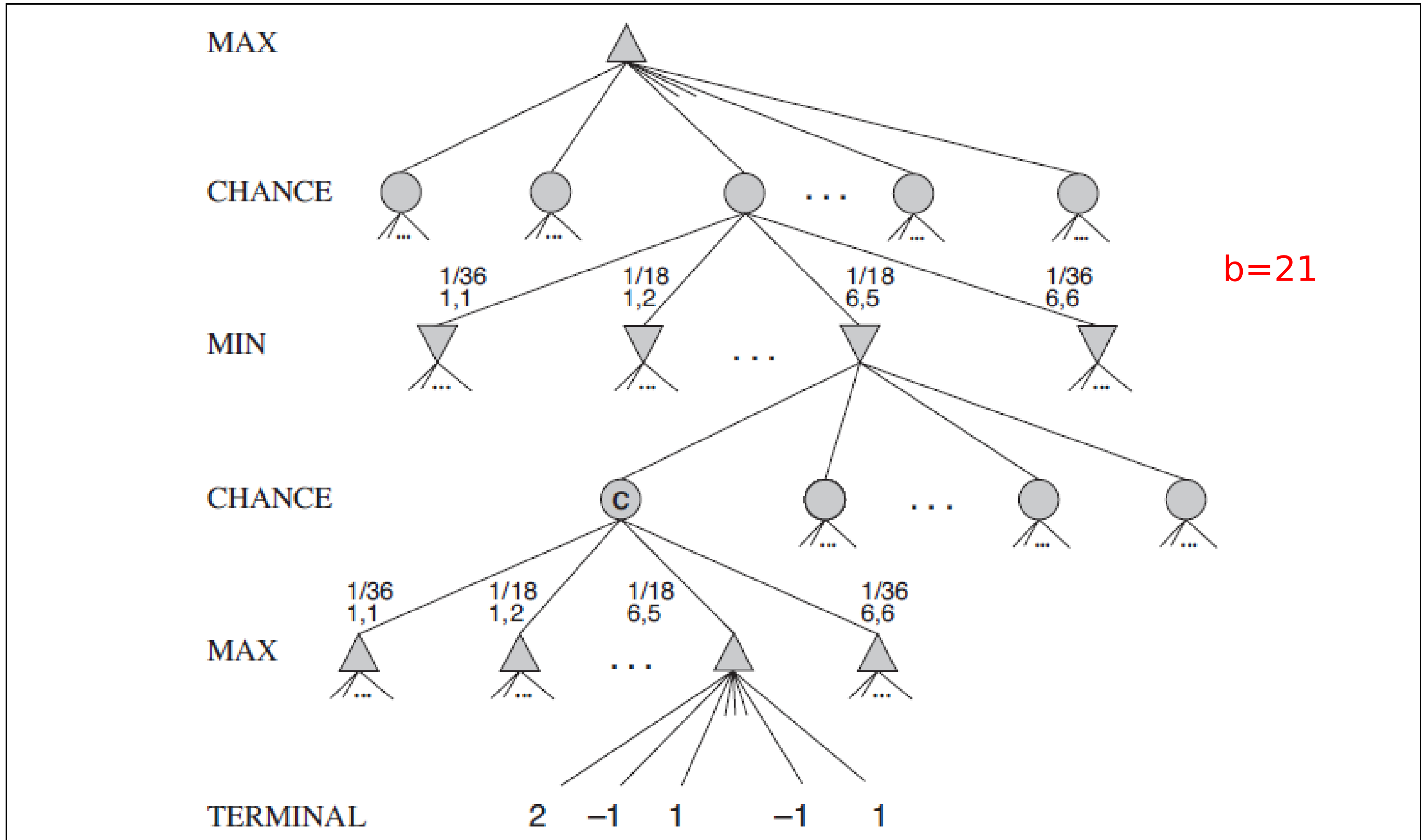
Nondeterministic games: backgammon



Combine luck and skill!
How is MIN-MAX tree handled?

Nondeterministic games: backgammon





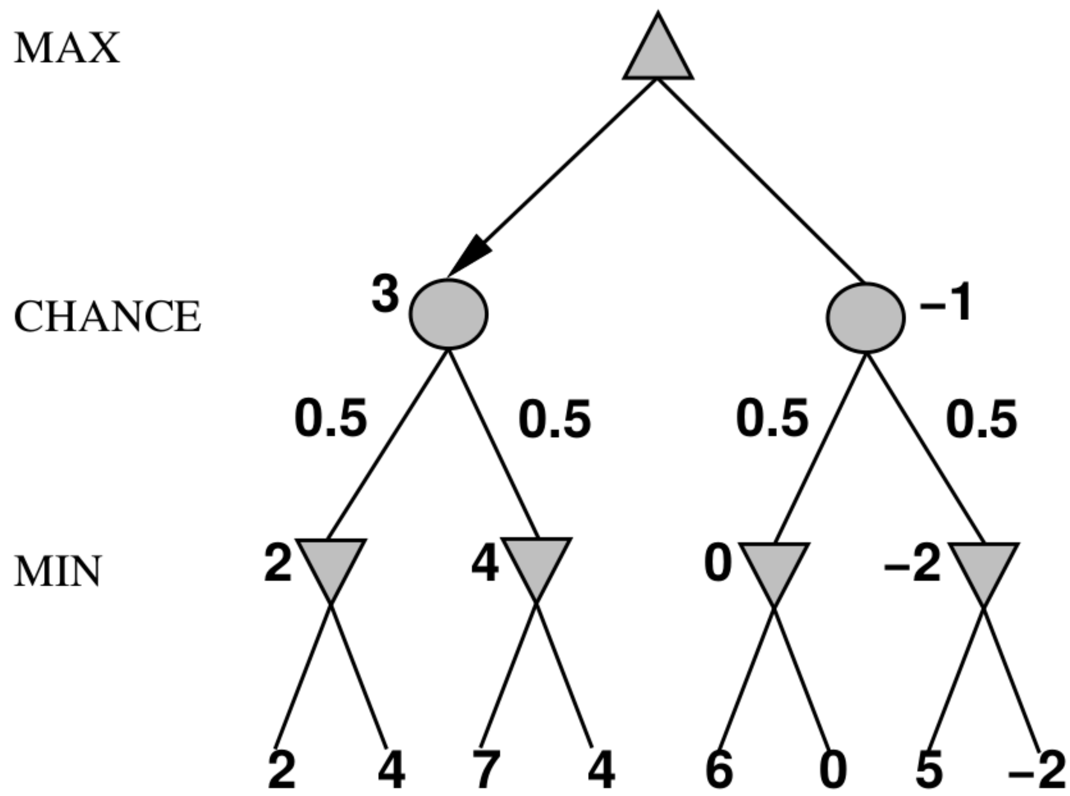
b=21

Figure 5.11 Schematic game tree for a backgammon position.

Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



how should we adapt the minimax algorithm?

max: return highest
min: return lowest
chance: ?

Algorithm for nondeterministic games

ExpectiMinimax gives perfect play

Just like Minimax, except we must also handle chance nodes:

. . .
if *state* is a Max node **then**
    return the highest ExpectiMinimax-Value of Successors(*state*)
if *state* is a Min node **then**
    return the lowest ExpectiMinimax-Value of Successors(*state*)
if *state* is a chance node **then**
    return average of ExpectiMinimax-Value of Successors(*state*)
. . .

Nondeterministic games in practice

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

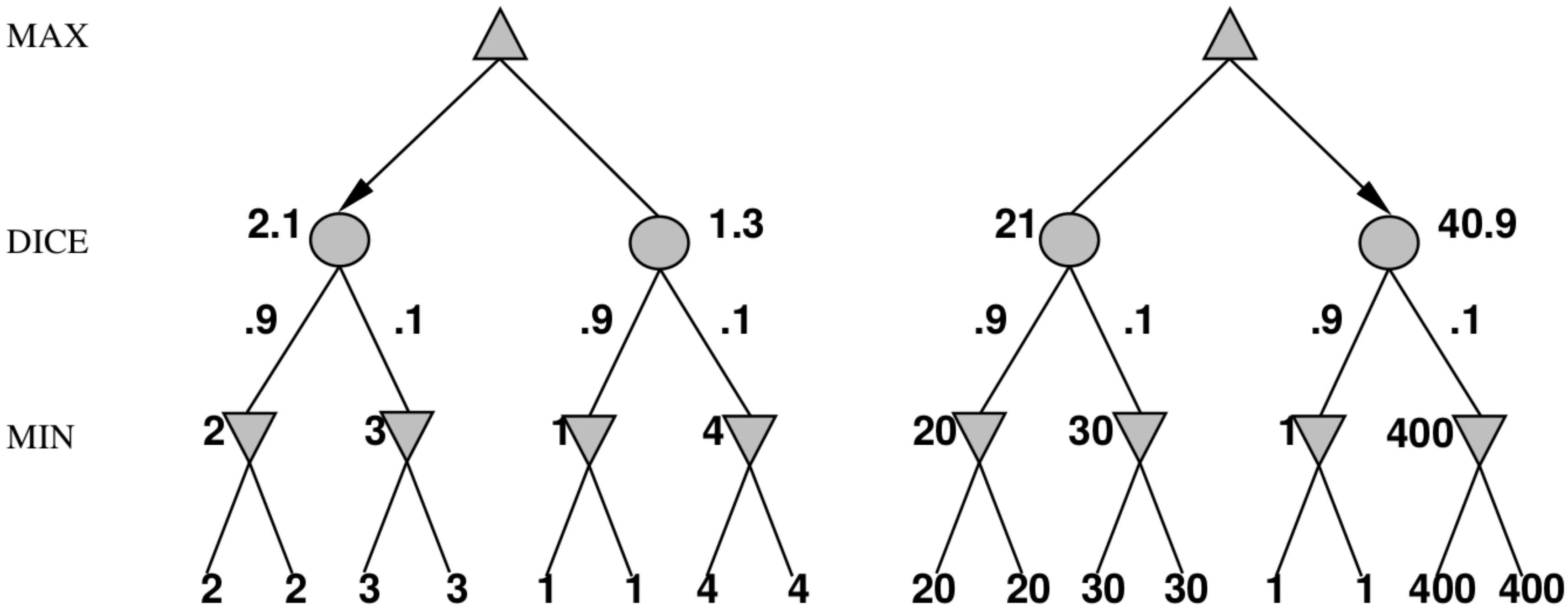
As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

TDGAMMON uses depth-2 search + very good EVAL
$\approx$ world-champion level

With minimax, satisfying the same ordering was sufficient.

Digression: Exact values DO matter



Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game*

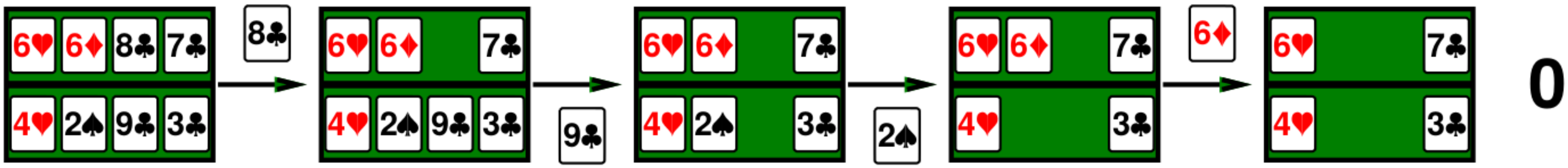
Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals*

Special case: if an action is optimal for all deals, it's optimal.*

GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

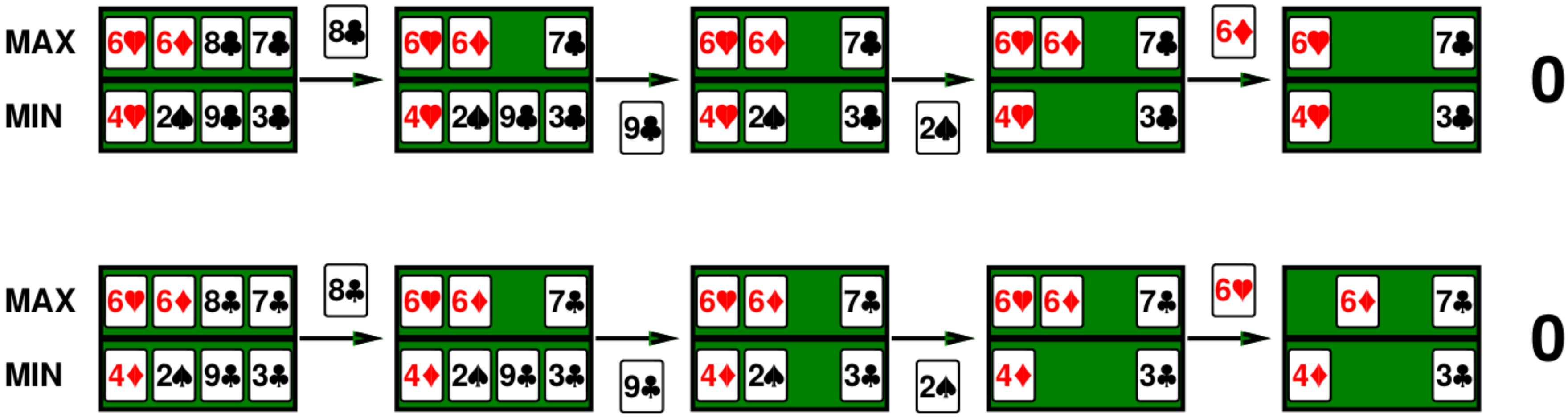
Example

Four-card bridge/whist/hearts hand, MAX to play first



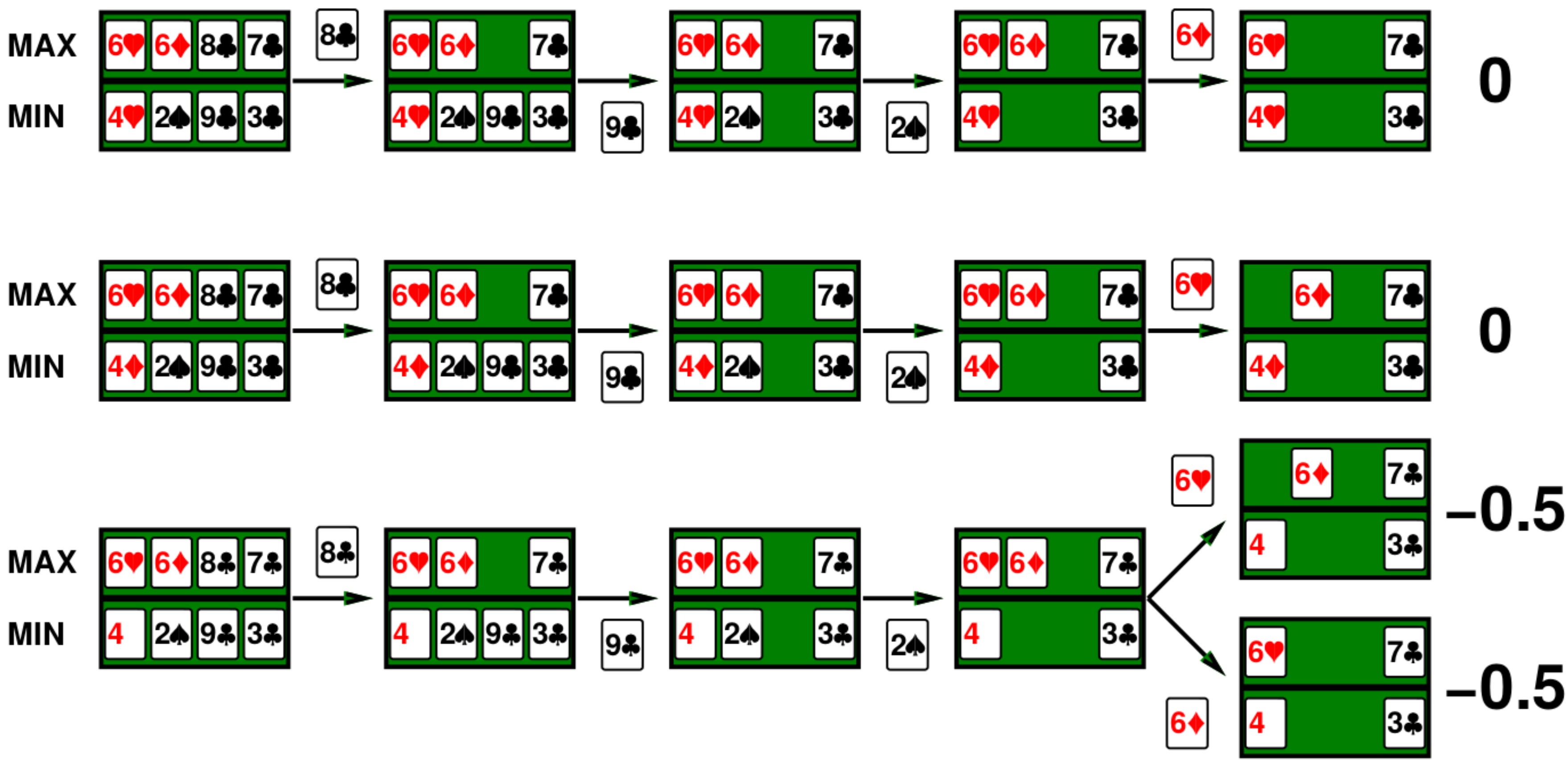
Example

Four-card bridge/whist/hearts hand, MAX to play first



Example

Four-card bridge/whist/hearts hand, MAX to play first



Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:

> take the left fork and you'll find a mound of jewels;
> take the right fork and you'll be run over by a bus.

Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you'll find a mound of jewels;
    take the right fork and you'll be run over by a bus.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you'll be run over by a bus;
    take the right fork and you'll find a mound of jewels.

Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you'll find a mound of jewels;
    take the right fork and you'll be run over by a bus.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you'll be run over by a bus;
    take the right fork and you'll find a mound of jewels.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    guess correctly and you'll find a mound of jewels;
    guess incorrectly and you'll be run over by a bus.

Proper analysis

* Intuition that the value of an action is the average of its values in all actual states is **WRONG**

With partial observability, value of an action depends on the information state or belief state the agent is in

Can generate and search a tree of information states

Leads to rational behaviors such as
- ◇ Acting to obtain information
- ◇ Signalling to one's partner
- ◇ Acting randomly to minimize information disclosure

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

◇ perfection is unattainable $\Rightarrow$ must approximate

◇ good idea to think about what to think about

◇ uncertainty constrains the assignment of values to states

◇ optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design

Constraint Satisfaction Problems

Outline

- $\diamondsuit$ CSP examples
- $\diamondsuit$ Backtracking search for CSPs
- $\diamondsuit$ Problem structure and problem decomposition
- $\diamondsuit$ Local search for CSPs

Constraint satisfaction problems (CSPs)

Standard search problem:

state is a "black box"—any old data structure
that supports goal test, eval, successor

CSP: states and goal test conform a standard, structured and simple representation.

state is defined by variables X_i with values from domain D_i

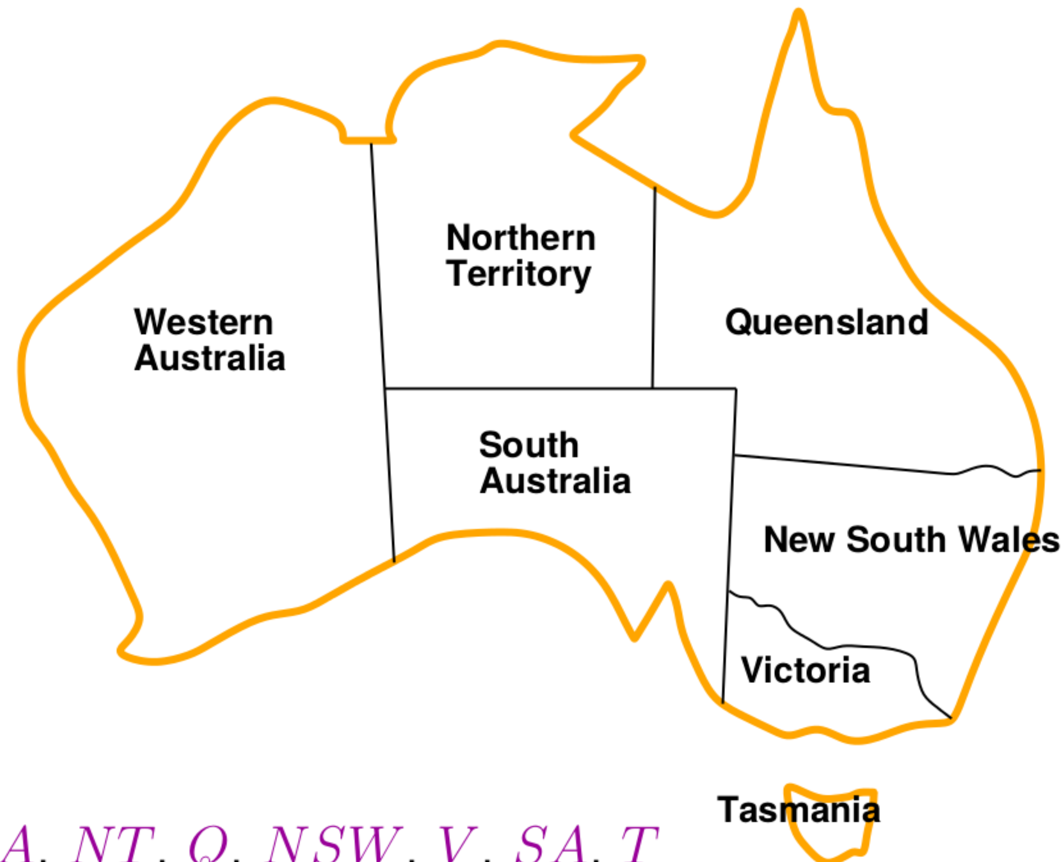
goal test is a set of constraints specifying
allowable combinations of values for subsets of variables

Simple example of a **formal representation language**

Allows useful **general-purpose** algorithms with more power than standard search algorithms

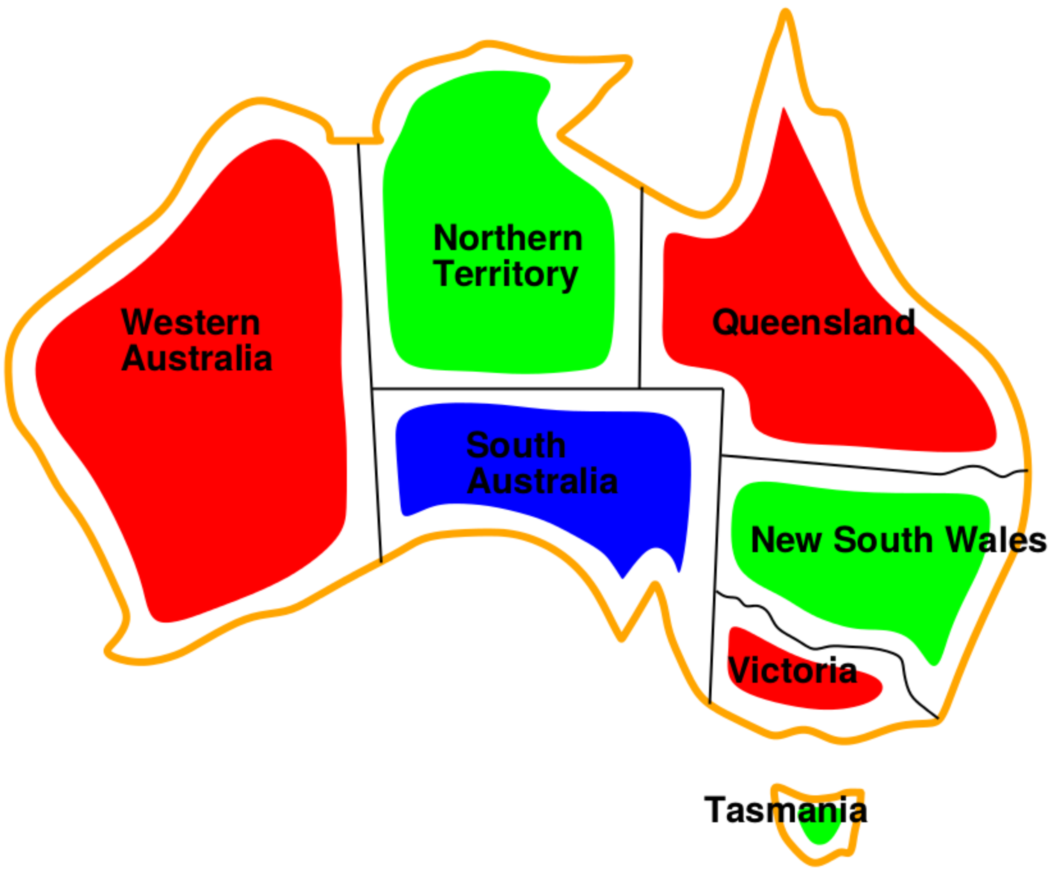
- state: assignment of variables {Xi=a, Xj=b..}
- assignment is consistent or legal if not violates constraints
- solution: a complete assignment that satisfies all constraints
- some CFPs require soln that maximize objective function

Example: Map-Coloring



Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring contd.



Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

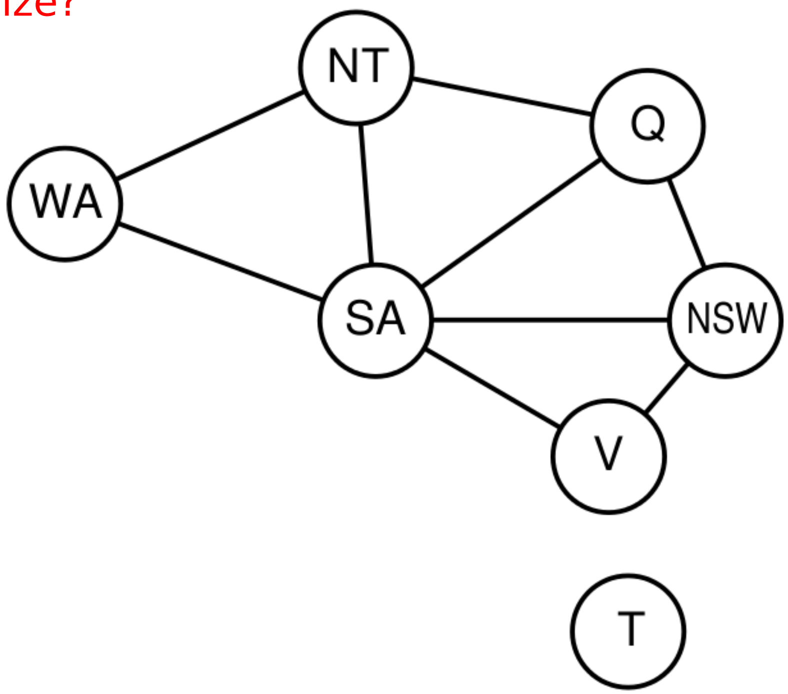
There are different solutions.

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints

Maybe easier to visualize?



General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Start from initial-state={}, assign a value in each step.

Varieties of CSPs

Discrete variables

e.g.? finite domains; size d $\Rightarrow$ $O(d^n)$ complete assignments why? Depth-first-search is popular. Why?

- ♢ e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)

infinite domains (integers, strings, etc.)

- ♢ e.g., job scheduling, variables are start/end days for each job
- ♢ need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- ♢ linear constraints solvable, nonlinear undecidable

enumerating assignments not possible

Continuous variables

- ♢ e.g., start/end times for Hubble Telescope observations
- ♢ linear constraints solvable in poly time by LP methods

Varieties of constraints

Unary constraints involve a single variable,
e.g., $SA \neq green$

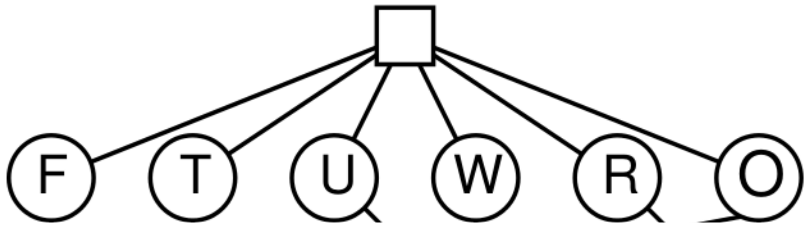
Binary constraints involve pairs of variables,
e.g., $SA \neq WA$

Higher-order constraints involve 3 or more variables,
e.g., cryptarithmetic column constraints

Preferences (soft constraints), e.g., red is better than $green$
often representable by a cost for each variable assignment
$\rightarrow$ constrained optimization problems

Example: Cryptarithmetic

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



Variables: $F\ T\ U\ W\ R\ O$

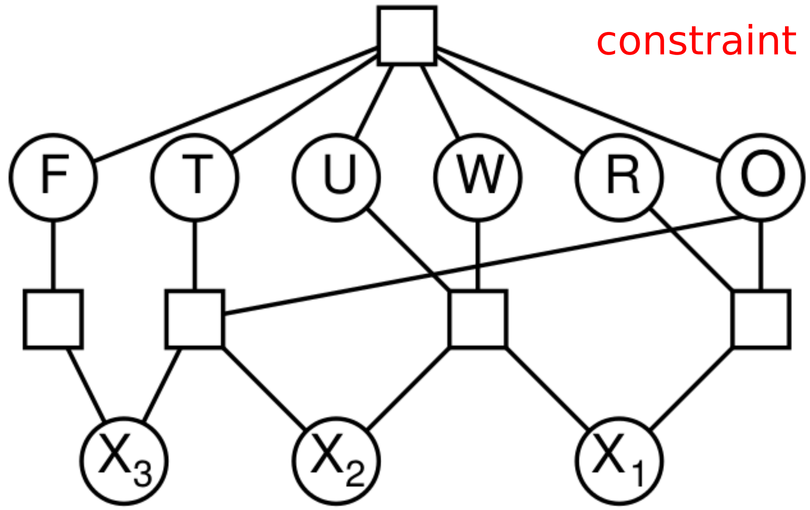
Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints

$\mathit{alldiff}(F, T, U, W, R, O)$ what else?

Example: Cryptarithmetic

```
    T W O
+   T W O
---------
  F O U R
```



constraint hypergraph

Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints

alldiff(F, T, U, W, R, O)

$O + O = R + 10 \cdot X_1$, etc.

Real-world CSPs

Assignment problems
    e.g., who teaches what class

Timetabling problems
    e.g., which class is offered when and where?

Hardware configuration

Spreadsheets

Absolute vs. preference constraints

Transportation scheduling

Factory scheduling

Floorplanning

Notice that many real-world problems involve real-valued variables

Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

◇ Initial state: the empty assignment, $\{\,\}$

◇ Successor function: assign a value to an unassigned variable
that does not conflict with current assignment.
$\Rightarrow$ fail if no legal assignments (not fixable!)

◇ Goal test: the current assignment is complete

1) This is the same for all CSPs! 😆
2) Every solution appears at depth n with n variables
$\Rightarrow$ use depth-first search d values
3) Path is irrelevant, so can also use complete-state formulation
4) How many leaves?

Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

◇ Initial state: the empty assignment, $\{ \}$

◇ Successor function: assign a value to an unassigned variable
that does not conflict with current assignment.
$\Rightarrow$ fail if no legal assignments (not fixable!)

◇ Goal test: the current assignment is complete

1) This is the same for all CSPs!
2) Every solution appears at depth n with n variables
$\Rightarrow$ use depth-first search
3) Path is irrelevant, so can also use complete-state formulation
4) $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!!

Backtracking search

Variable assignments are commutative, i.e.,

$[WA = red$ then $NT = green]$ same as $[NT = green$ then $WA = red]$

Only need to consider assignments to a single variable at each node

$\Rightarrow$ $b = d$ and there are d^n leaves

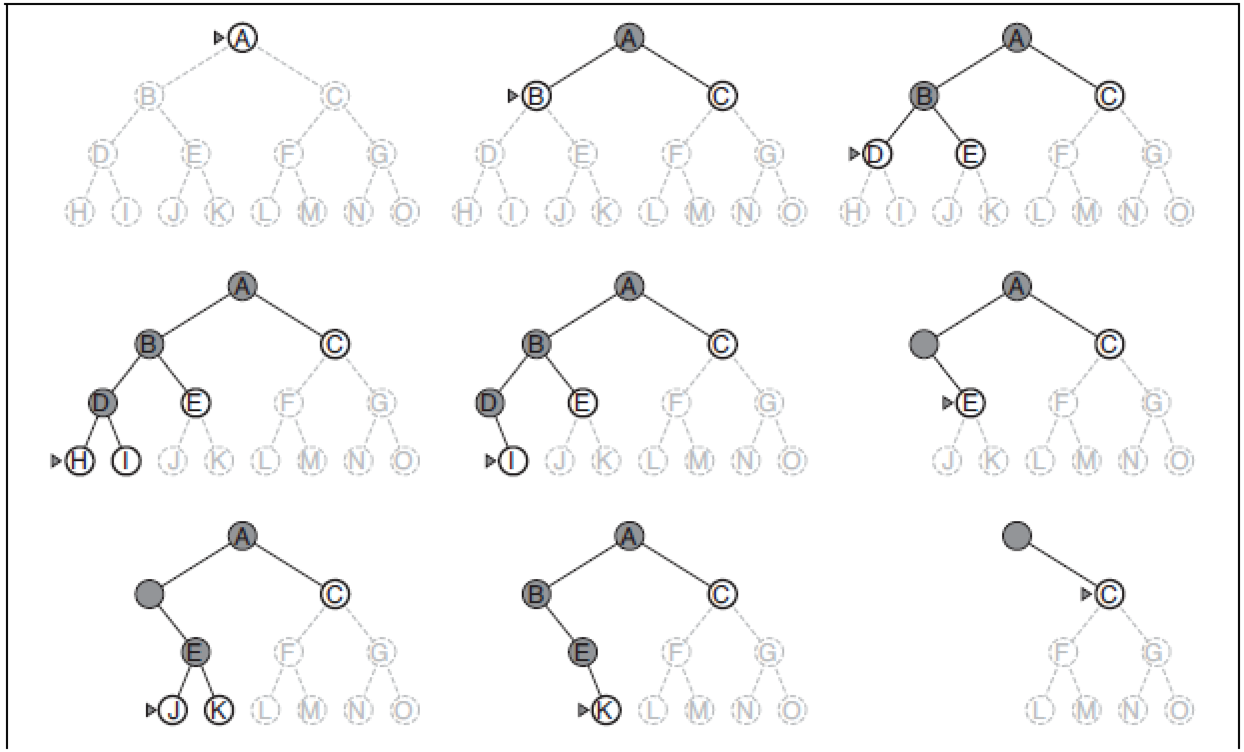
Depth-first search for CSPs with single-variable assignments
is called backtracking search

Backtracking search is the basic uninformed algorithm for CSPs

Can solve n-queens for $n \approx 25$

Depth-first-search: Backtracking

- A variant of depth-first search called backtracking BACKTRACKING search uses still less memory.
- Only one successor is generated at a time rather than all successors;
- Each partially expanded node remembers which successor to generate next.
- In this way, only O(m) memory is needed rather than O(bm).



Backtracking search

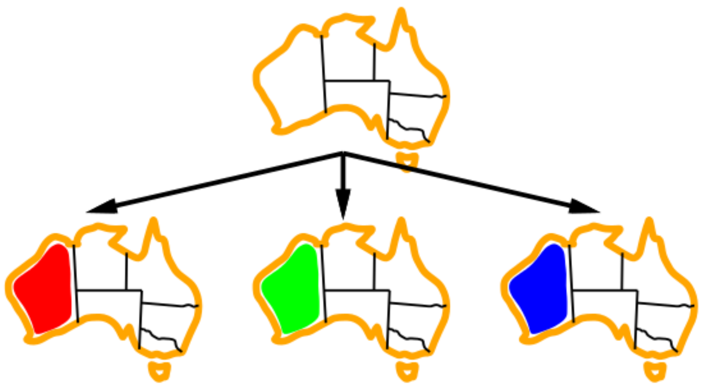
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
       if value is consistent with assignment given CONSTRAINTS[csp] then
           add {var = value} to assignment
           result ← RECURSIVE-BACKTRACKING(assignment, csp)
           if result ≠ failure then return result
           remove {var = value} from assignment
   return failure
```

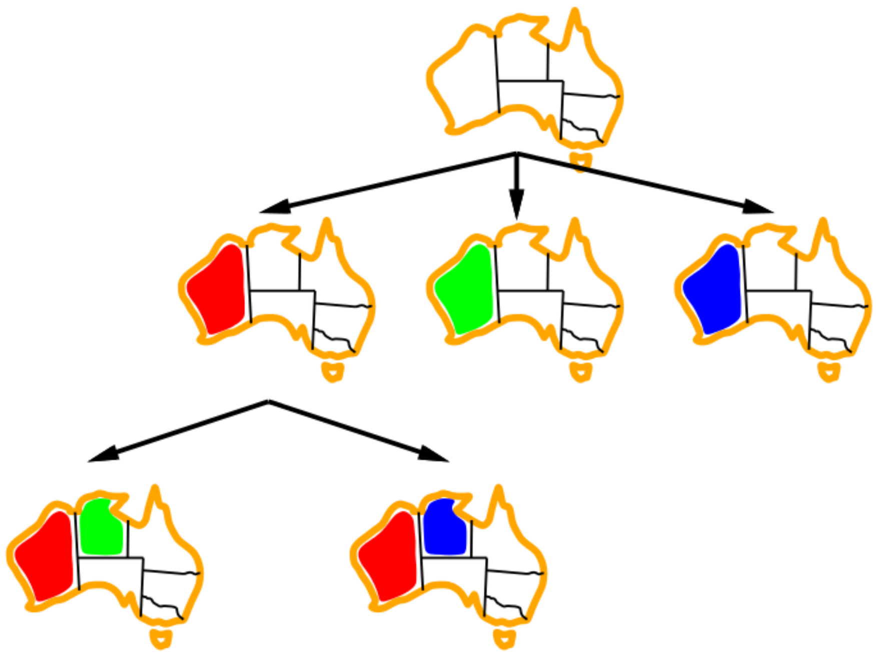
add backtracking search from pg. 76

Backtracking example

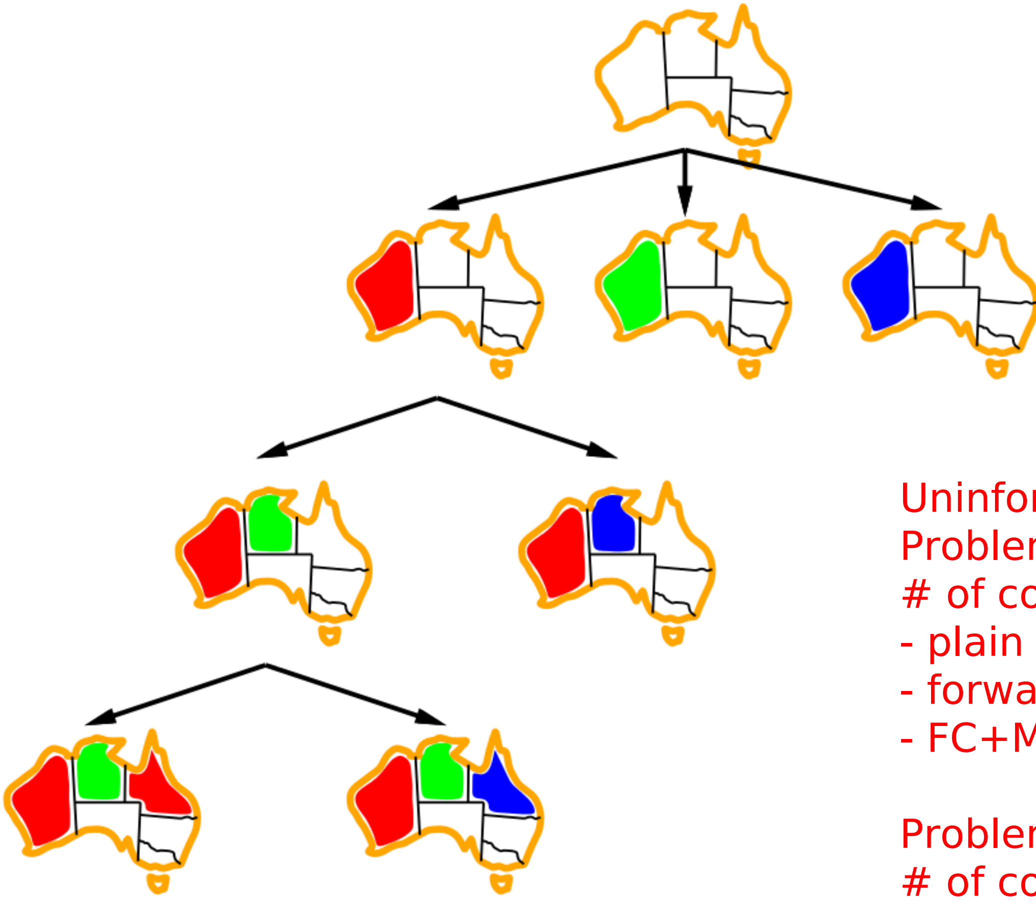
Backtracking example



Backtracking example



Backtracking example



Uninformed algorithm. No big expectations
Problem: 4-coloring of 50 USA states
# of consistency checks:
- plain backtracking: >1,000K
- forward checking: >1,000K
- FC+MRV: 60

Problem: n-queens
# of consistency checks:
- plain backtracking: >40,000K
- forward checking: >40,000K
- FC+MRV: 717K

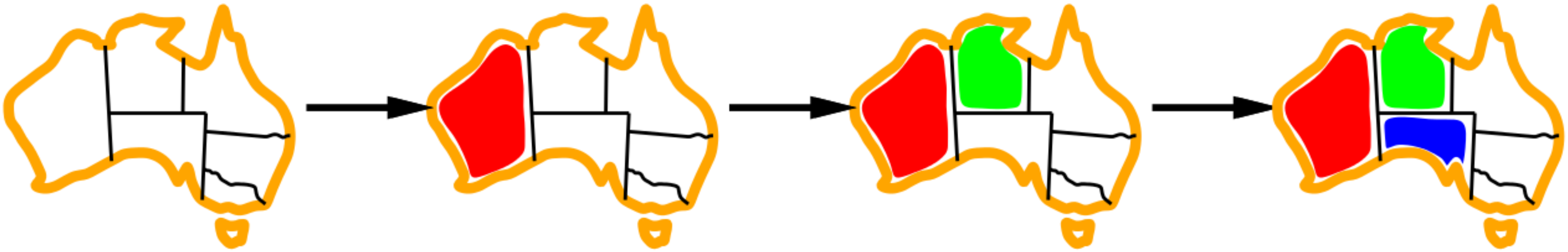
Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

Minimum remaining values

Minimum remaining values (MRV):
choose the variable with the fewest legal values



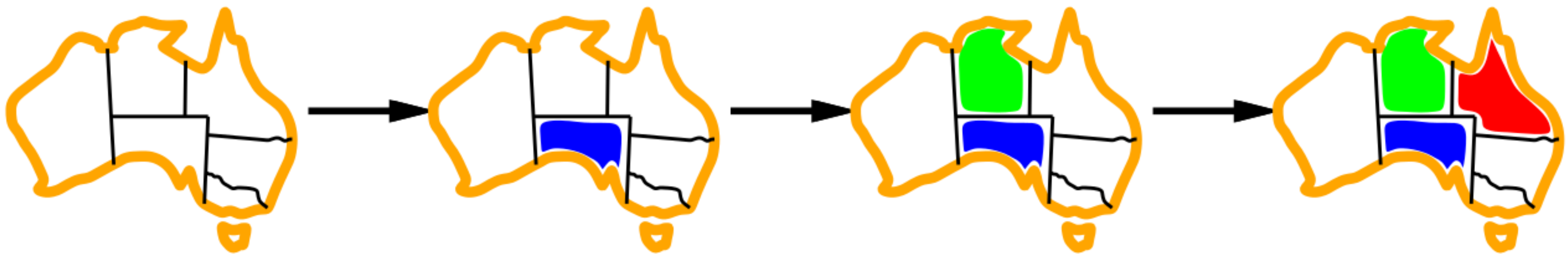
called "the most constrained variable"

Uninformed algorithm. No big expectations
Problem: n-queens
# of consistency checks:
- plain backtracking: >40,000K
- BT+MRV > 13,500K
- forward checking: >40,000K
- FC+MRV: 717K

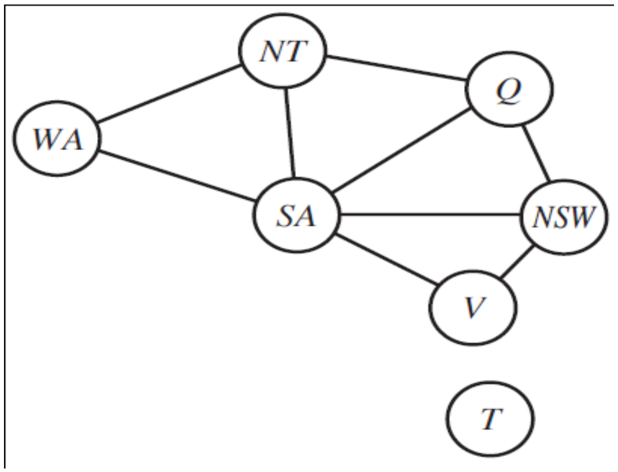
Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:
choose the variable with the most constraints on remaining variables



Attempt to reduce branching factor of future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.

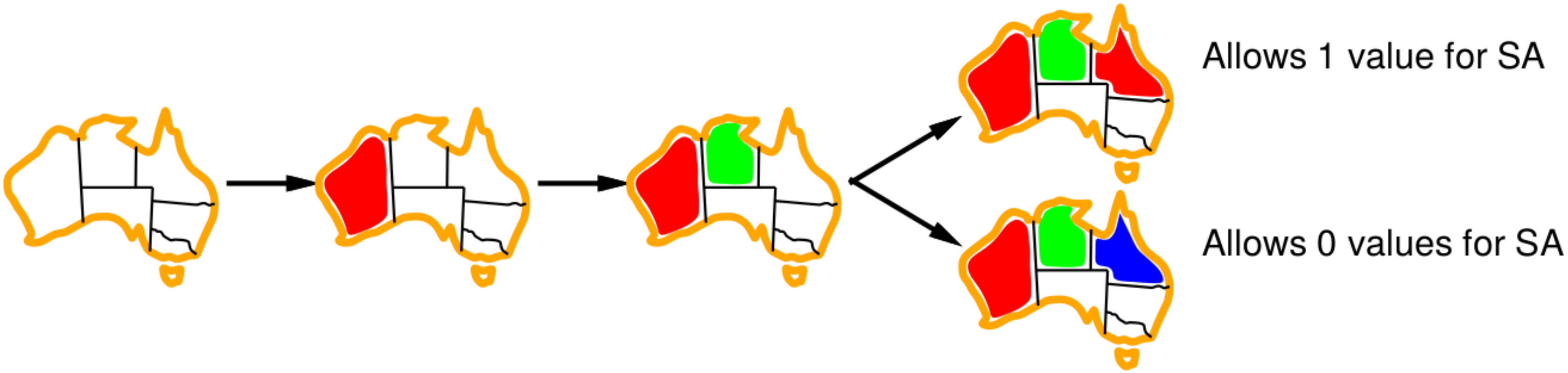


Least constraining value

Given a variable, choose the least constraining value:

the one that rules out the fewest values in the remaining variables

i.e. leave the max flexibility for subsequent variable assignments.



Combining these heuristics makes 1000 queens feasible

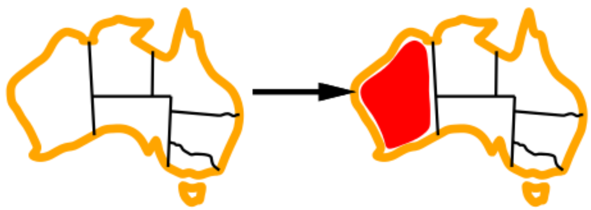
So far, we only considered the constraints on a variable only at a time

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

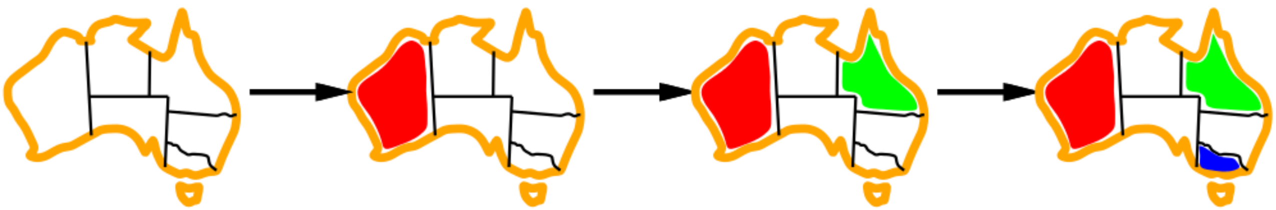


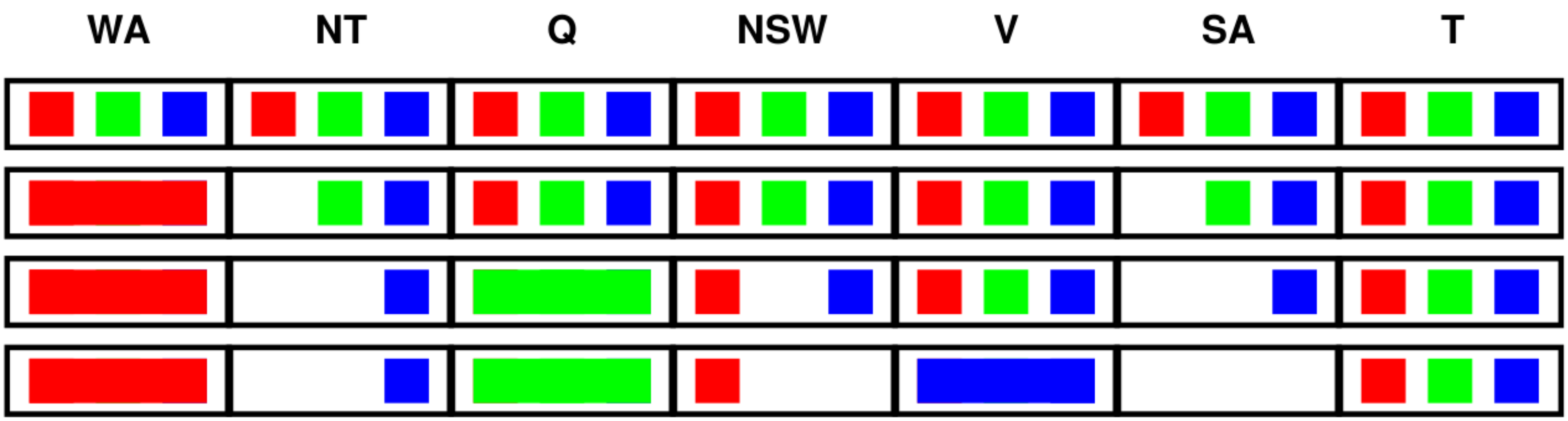
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values





But probably we would select either NT or SA

Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

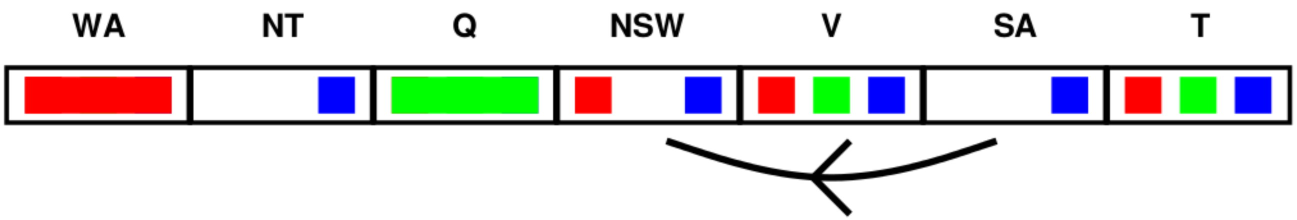
general term for propagating the implications of a constraint on one variable onto other variables.

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y

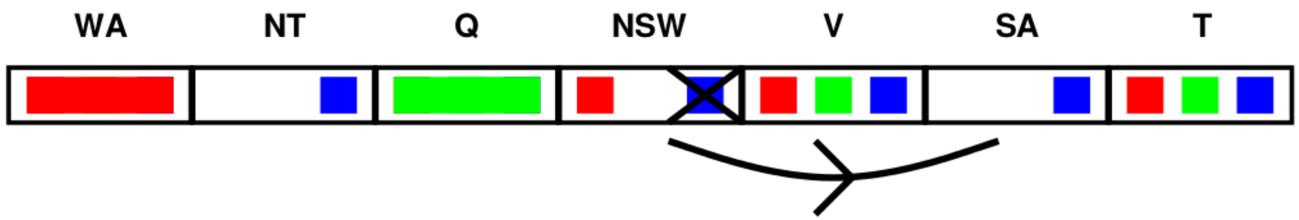


Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y

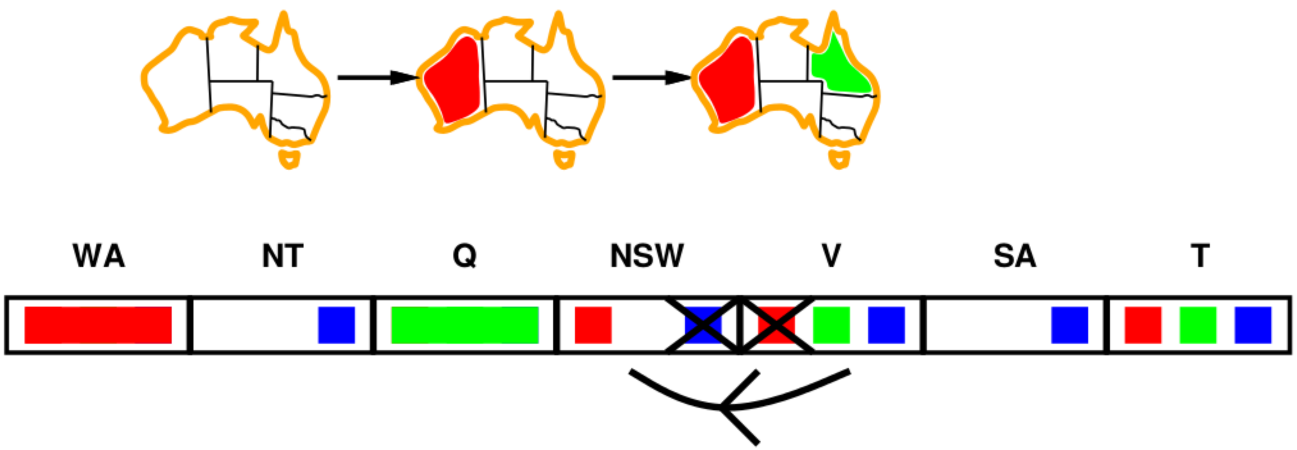


Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y

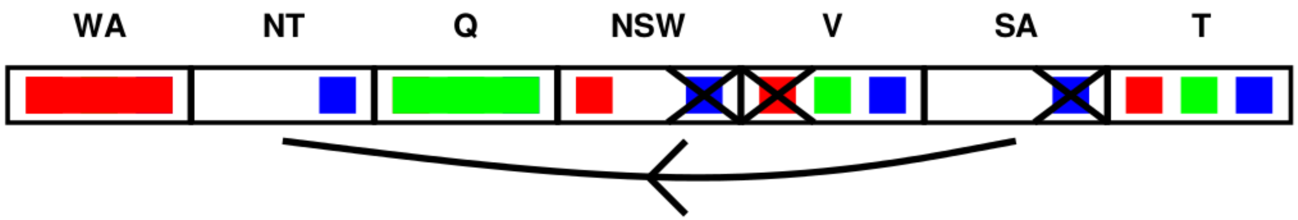


If X loses a value, neighbors of X need to be rechecked

Arc consistency

Simplest form of propagation makes each arc consistent

$X \to Y$ is consistent iff
for **every** value x of X there is **some** allowed y



If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Arc consistency algorithm

```
function AC-3(csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (X_i, X_j) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
         for each X_k in NEIGHBORS[X_i] do
            add (X_k, X_i) to queue
```

```
function REMOVE-INCONSISTENT-VALUES(X_i, X_j) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[X_i] do
      if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint X_i ↔ X_j
         then delete x from DOMAIN[X_i];  removed ← true
   return removed
```

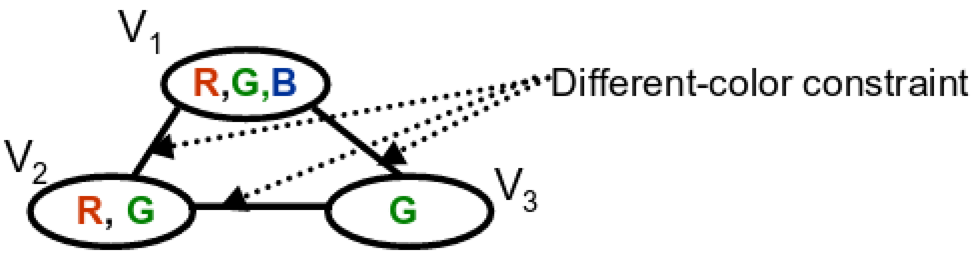
$O(n^2 d^3)$ (but detecting **all** is NP-hard)

^ compute!

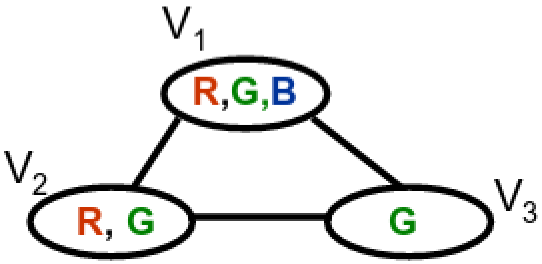
Constraint Propagation Example

Graph Coloring

Initial Domains are indicated



Arc examined	Value deleted



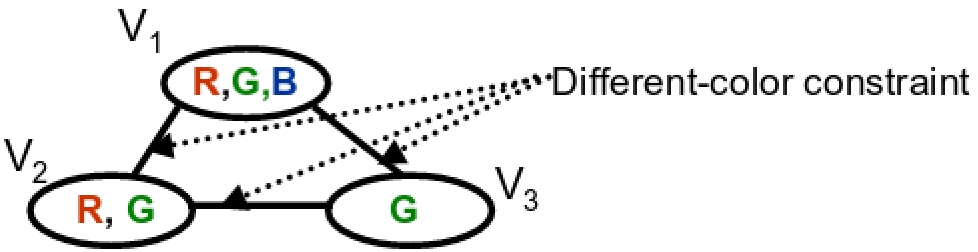
Each undirected constraint arc is really two directed constraint arcs, the effects shown above are from examining BOTH arcs.

http://web.mit.edu/6.034/wwwbob/constraint.pdf

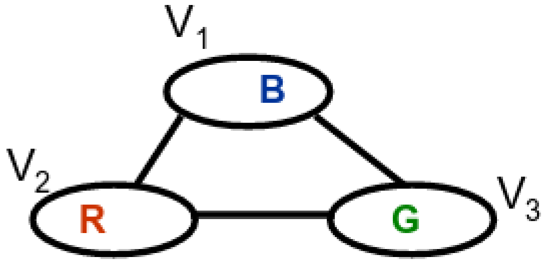
Constraint Propagation Example

Graph Coloring

Initial Domains are indicated



Arc examined	Value deleted
$V_1 - V_2$	none
$V_1 - V_3$	V_1(G)
$V_2 - V_3$	V_2(G)
$V_1 - V_2$	V_1(R)
$V_1 - V_3$	none
$V_2 - V_3$	none

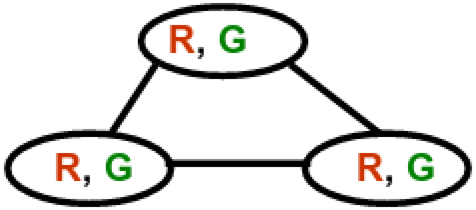


http://web.mit.edu/6.034/wwwbob/constraint.pdf

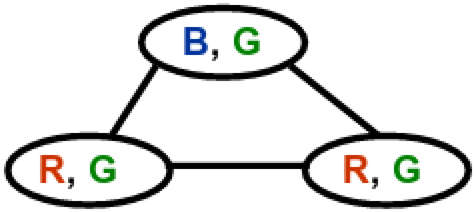
Constraint Propagation Example

But, arc consistency is not enough in general

Graph Coloring

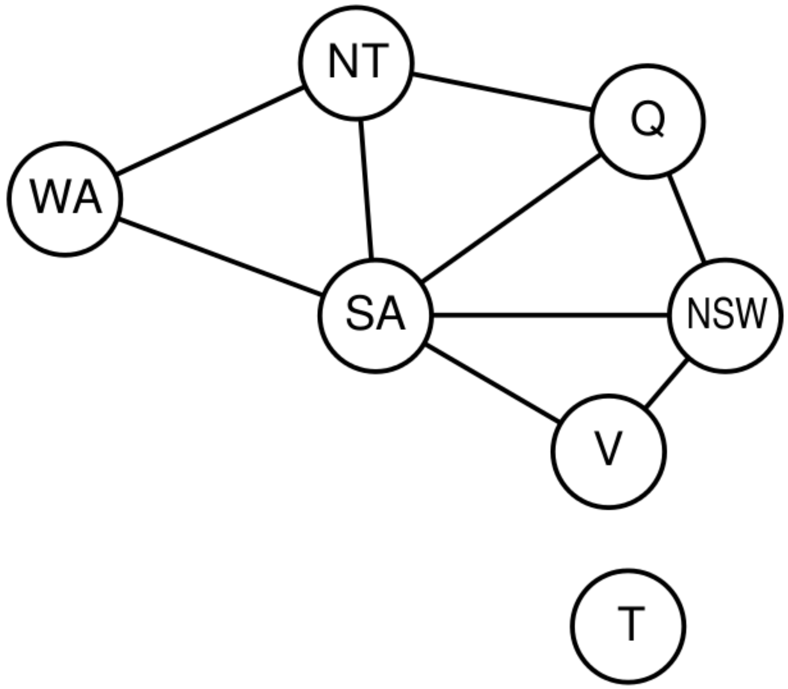


arc consistent but
no solutions



arc consistent but 2
solutions B,R,G ;
B,G,R .

Problem structure



Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph

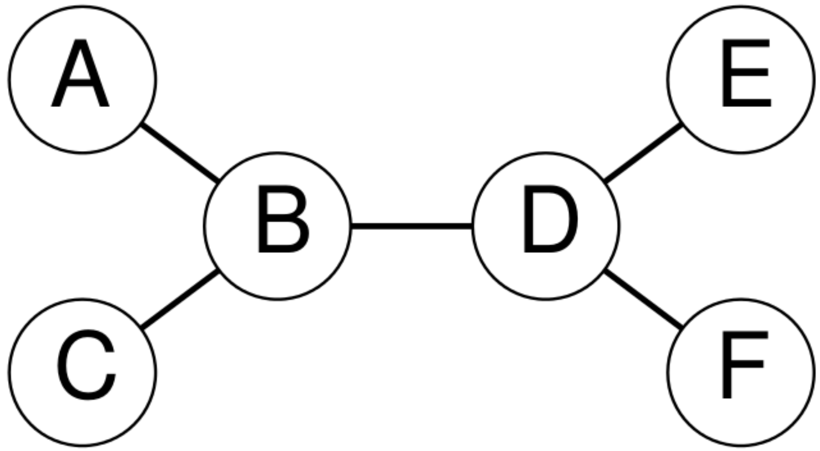
Problem structure contd.

Suppose each subproblem has c variables out of n total

Worst-case solution cost is $n/c \cdot d^c$, **linear** in n

E.g., $n = 80$, $d = 2$, $c = 20$
2^{80} = 4 billion years at 10 million nodes/sec
$4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec

Tree-structured CSPs



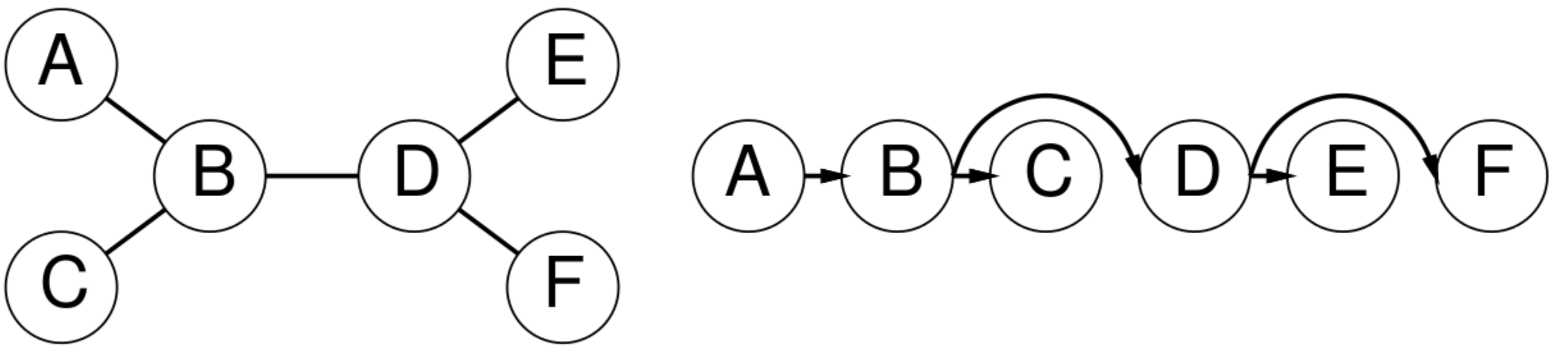
Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time

Compare to general CSPs, where worst-case time is $O(d^n)$

This property also applies to logical and probabilistic reasoning:
an important example of the relation between syntactic restrictions
and the complexity of reasoning.

Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



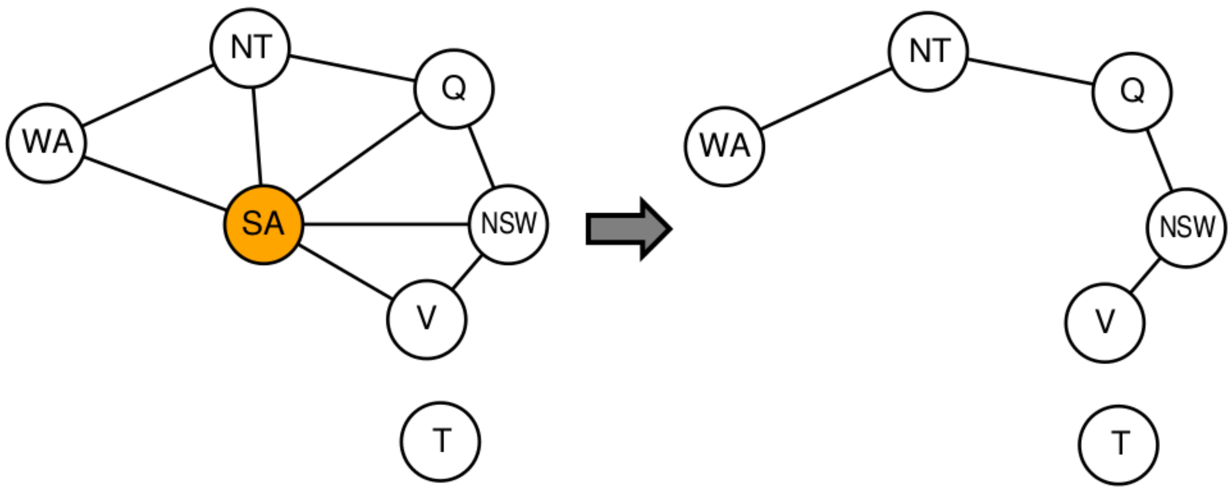
2. For j from n down to 2, apply $\textsc{RemoveInconsistent}(Parent(X_j), X_j)$

why do we remove
in backwards order?

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Nearly tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbors' domains



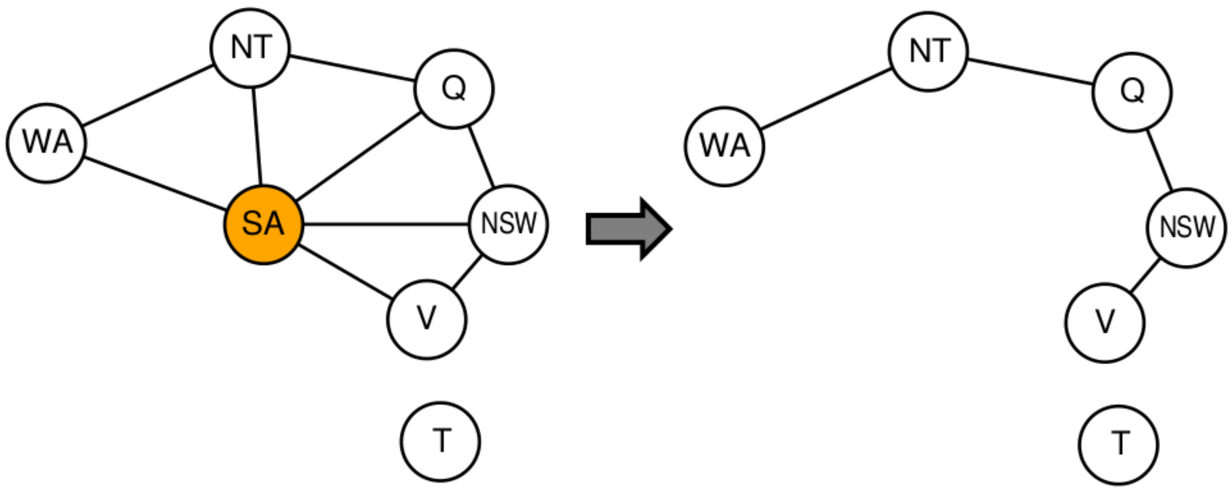
Cutset conditioning: instantiate (in all ways) a set of variables
such that the remaining constraint graph is a tree

Cutset size c $\Rightarrow$ runtime $O($???? $)$, very fast for small c

1. Choose a subset S of variables such that constraint graph becomes tree.
2. For each possible assignment of S that satisfies all constraints on S
   (a) remove from domains of the remaining variables that are inconsistent with assign. of S
   (b) if remaining CSP has a solution..

Nearly tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size c $\Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c

1. Choose a subset S of variables such that constraint graph becomes tree.
2. For each possible assignment of S that satisfies all constraints on S
   (a) remove from domains of the remaining variables that are inconsistent with assign. of S
   (b) if remaining CSP has a solution..

Iterative algorithms for CSPs

Hill-climbing, simulated annealing typically work with
"complete" states, i.e., all variables assigned

To apply to CSPs:
allow states with unsatisfied constraints
operators **reassign** variable values

Variable selection: randomly select any conflicted variable

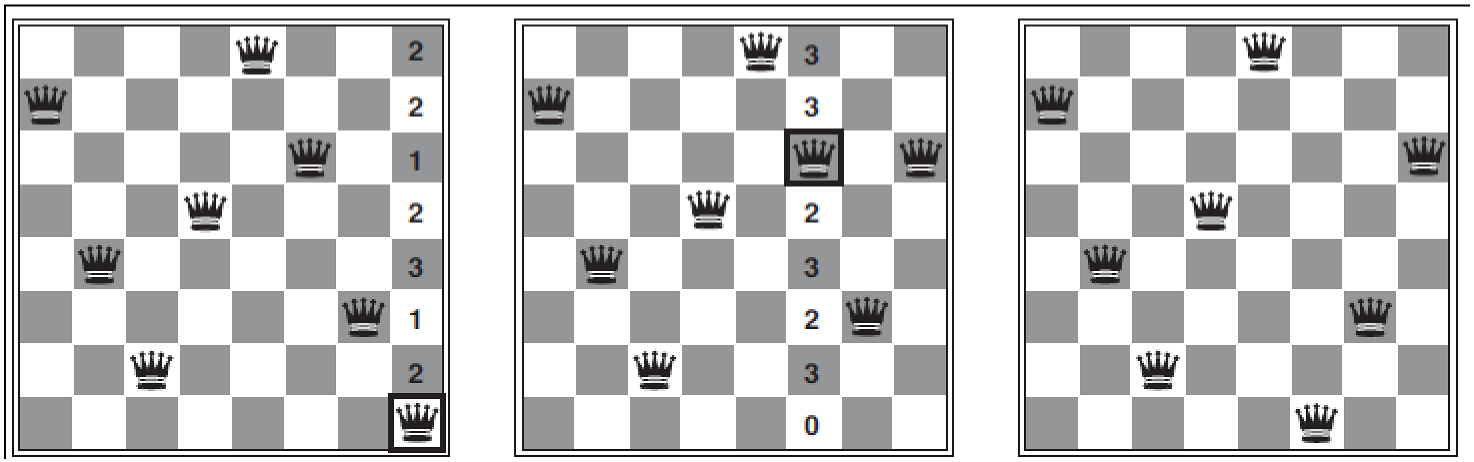
Value selection by min-conflicts heuristic:
choose value that violates the fewest constraints
i.e., hillclimb with $h(n)$ = total number of violated constraints

Local search for CSP

- Min-conflicts heuristic: select the value that results in min number of conflicts with other variables. Surprisingly effective

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
   inputs: csp, a constraint satisfaction problem
           max_steps, the number of steps allowed before giving up

   current ← an initial complete assignment for csp
   for i = 1 to max_steps do
      if current is a solution for csp then return current
      var ← a randomly chosen conflicted variable from csp.VARIABLES
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
   return failure
```



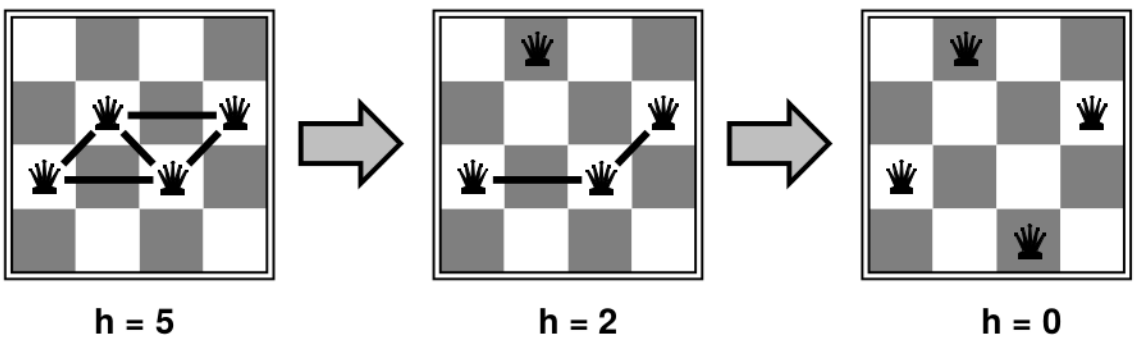
Example: 4-Queens

States: 4 queens in 4 columns ($4^4 = 256$ states)

Operators: move queen in column

Goal test: no attacks

Evaluation: $h(n) =$ number of attacks



Summary

CSPs are a special kind of problem:
states defined by values of a fixed set of variables
goal test defined by constraints on variable values

Backtracking = depth-first search with one variable assigned per node

Variable ordering and value selection heuristics help significantly

Forward checking prevents assignments that guarantee later failure

Constraint propagation (e.g., arc consistency) does additional work
to constrain values and detect inconsistencies

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time

Iterative min-conflicts is usually effective in practice