Artificial Intelligence

AI

- A serious science.
- General-purpose AI like the robots of science fiction is incredibly hard
  - Human brain appears to have lots of special and general functions, integrated in some amazing way that we really do not understand at all (yet)
- Special-purpose AI E.g., chess/poker playing programs, logistics planning, automated translation, voice recognition, web search, data mining, medical diagnosis, keeping a car on the road

Definitions of AI

- Systems that think like humans
- Systems that think rationally
- Systems that act like humans
- Systems that act rationally

Turing Test

- (Human) judge communicates with a human and a machine over text-only channel,
- Both human and machine try to act like a human,
- Judge tries to tell which is which.
- Numerous variants
- Current programs nowhere close to passing this
- It is possible to (temporarily) fool humans who do not realize they may be talking to a robot

History of AI

- 50s/60s: Early successes! AI can draw logical conclusions, prove some theorems, create simple plans… Some initial work on neural networks…
- Led to overhyping: researchers promised funding agencies spectacular progress, but started running into difficulties:
  - **Ambiguity**: highly funded translation programs (Russian to English) were good at syntactic manipulation but bad at disambiguation
  - **Scalability/complexity**: early examples were very small, programs could not scale to bigger instances

70s, 80s: Creation of expert systems (systems specialized for one particular task based on experts' knowledge), wide industry adoption

Searching methods

- We have some actions that can change the state of the world
  - Change induced by an action perfectly predictable
- Try to come up with a sequence of actions that will lead us to a goal state
  - May want to minimize number of actions
  - More generally, may want to minimize total cost of actions
- Do not need to execute actions in real life while searching for solution!
  - Everything perfectly predictable anyway

Key concepts in search

- Set of states that we can be in
  - Including an initial state…
  - … and goal states (equivalently, a goal test)
- For every state, a set of actions that we can take
  - Each action results in a new state
  - Typically defined by successor function
    - Given a state, produces all states that can be reached from it
- Cost function that determines the cost of each action (or path = sequence of actions)
- Solution: path from initial state to a goal state
  - Optimal solution: solution with minimal cost

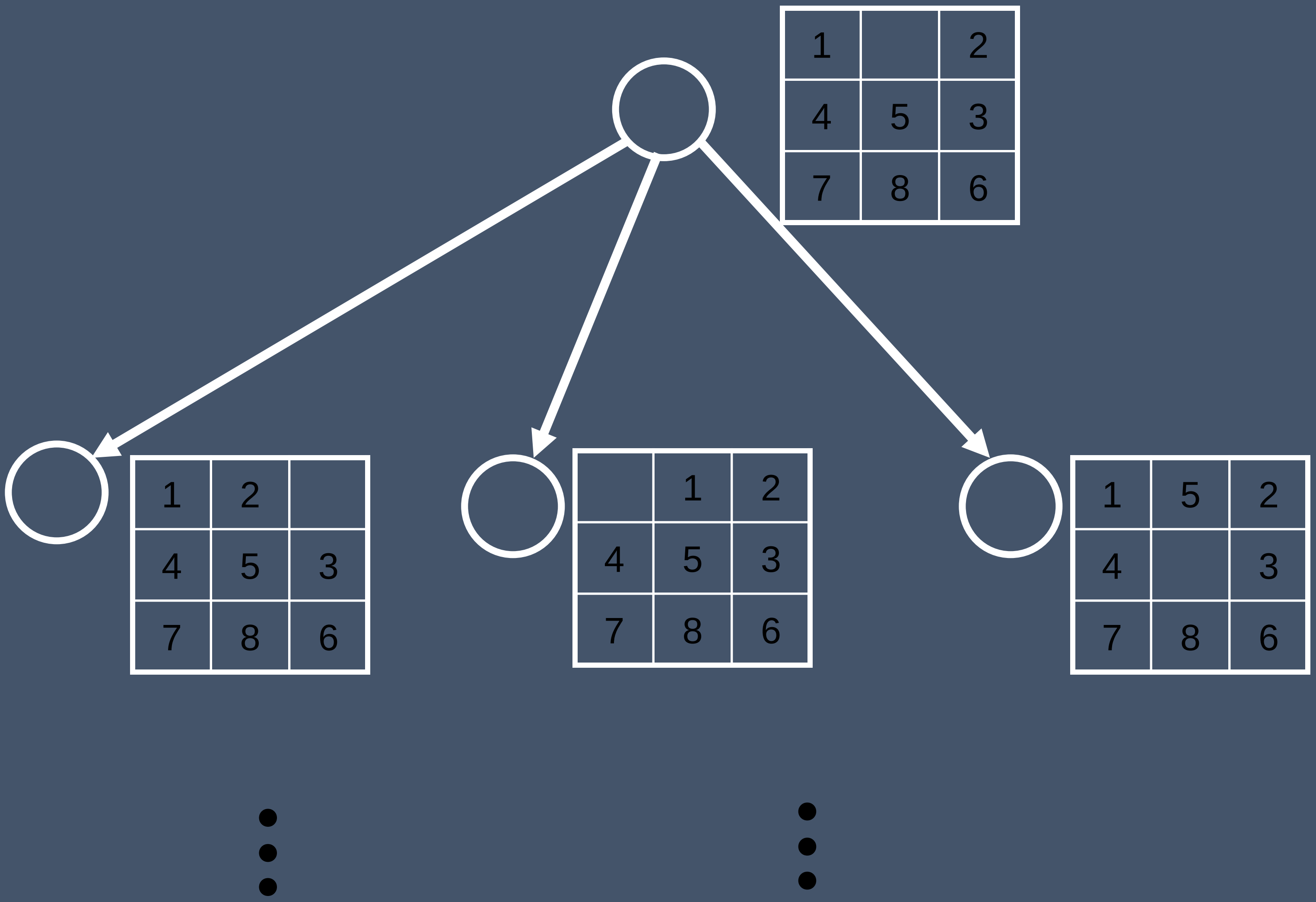
8-puzzle

1		2
4	5	3
7	8	6

1	2	3
4	5	6
7	8	

goal state

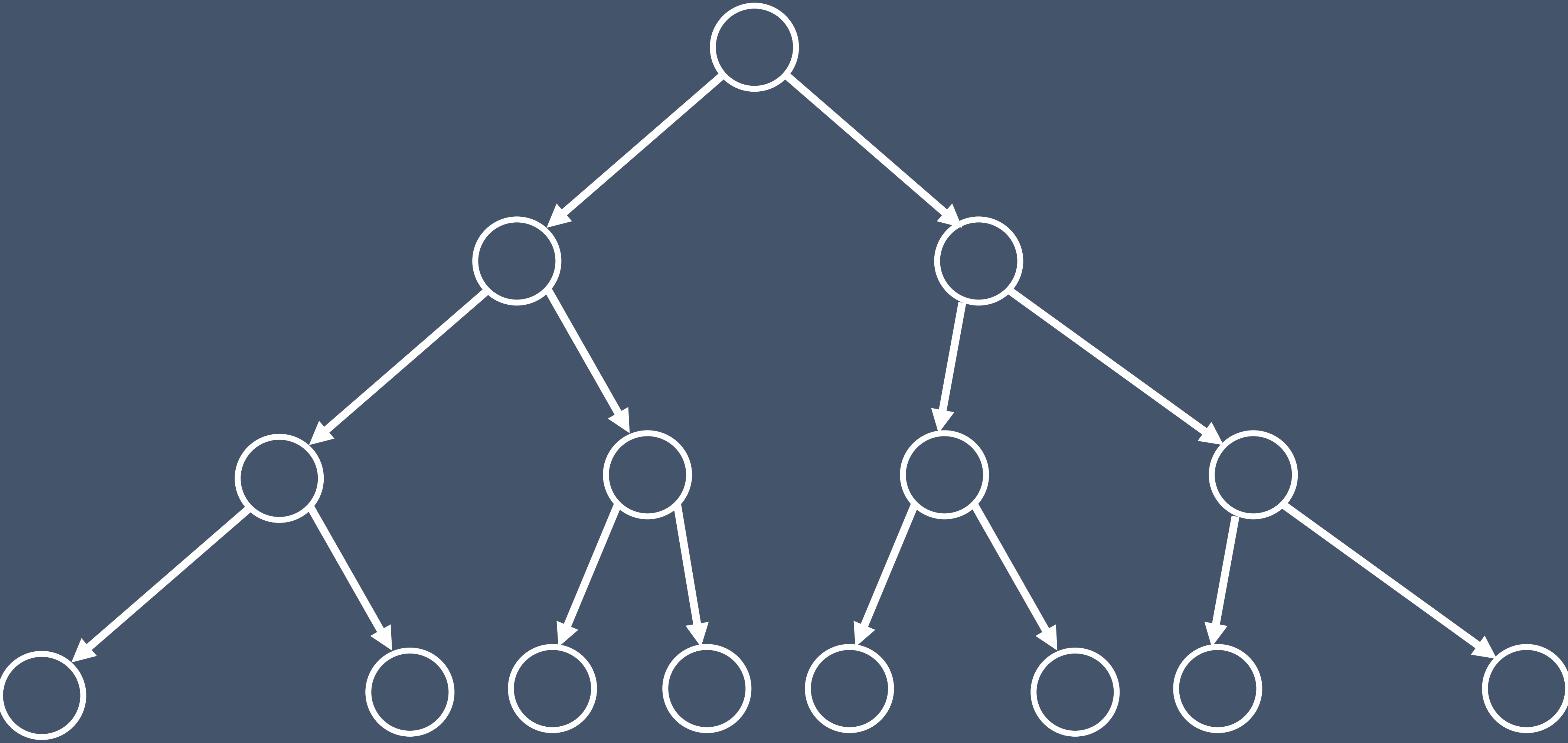
8-puzzle



Uninformed Search methods

- Given a state, we only know whether it is a goal state or not
- Cannot say one nongoal state looks better than another nongoal state
- Can only traverse state space blindly in hope of somehow hitting a goal state at some point
    - Also called blind search
    - Blind does **not** imply unsystematic!

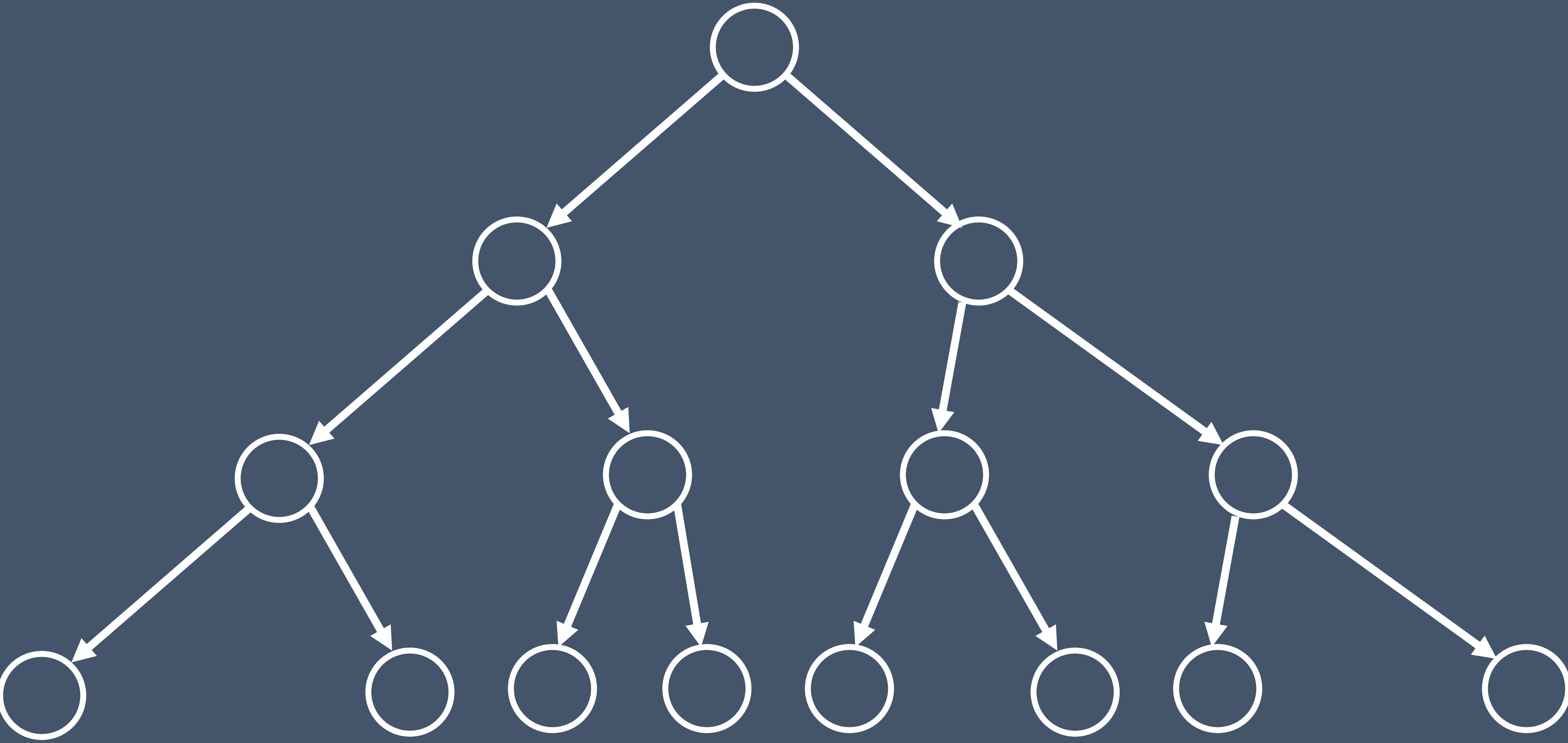
Breadth-first search



Properties of breadth-first search

- Nodes are expanded in the same order in which they are generated
  - Fringe can be maintained as a First-In-First-Out (FIFO) queue
- BFS is complete: if a solution exists, one will be found
- BFS finds a shallowest solution
  - Not necessarily an optimal solution
- If every node has b successors (the branching factor), first solution is at depth d, then fringe size will be at least b^d at some point
  - This much space (and time) required

Depth-first search



Implementing depth-first search

- Fringe can be maintained as a Last-In-First-Out (LIFO) queue (aka. a stack)
- Also easy to implement recursively:
- DFS(node)
    - If goal(node) return solution(node);
    - For each successor of node
        - Return DFS(successor) unless it is *failure*;
    - Return *failure;*

Properties of depth-first search

- Not complete (might cycle through nongoal states)
- If solution found, generally not optimal/shallowest
- If every node has b successors (the branching factor), and we search to at most depth m, fringe is at most bm
  - Much better space requirement ☺
  - Actually, generally don't even need to store all of fringe
- Time: still need to look at every node
  - $b^m + b^{m-1} + \ldots + 1$ (for $b>1$, $O(b^m)$)
  - Inevitable for uninformed search methods…

BFS and DFS

- Limited depth DFS: just like DFS, except never go deeper than some depth d
- Iterative deepening DFS:
  - Call limited depth DFS with depth 0;
  - If unsuccessful, call with depth 1;
  - If unsuccessful, call with depth 2;
  - Etc.
- Complete, finds shallowest solution
- Space requirements of DFS
- May seem wasteful timewise because replicating effort
  - Really not that wasteful because **almost all effort at deepest level**
  - $db + (d-1)b^2 + (d-2)b^3 + \ldots + 1b^d$ is $O(b^d)$ for $b > 1$

Uniform Cost

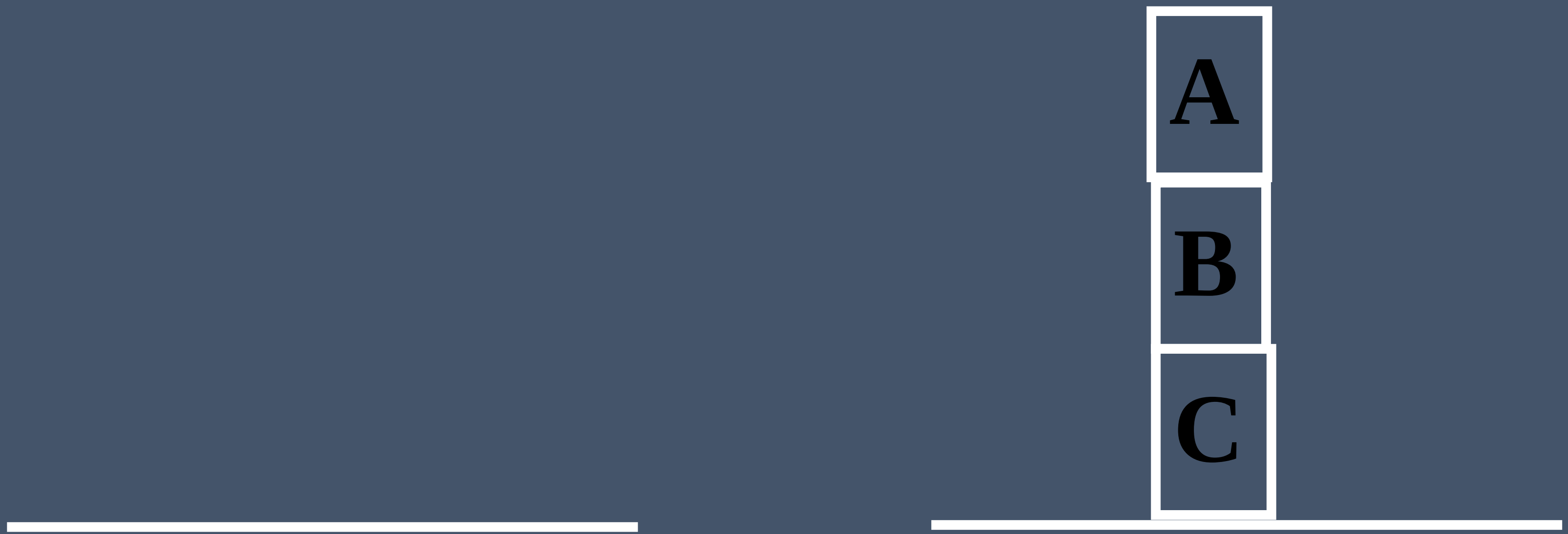
- BFS finds shallowest solution because always works on shallowest nodes first
- Similar idea: always work on the lowest-cost node first (uniform-cost search)
- Will find optimal solution (assuming costs increase by at least constant amount along path)
- Will often pursue lots of short steps first
- If optimal cost is C, and cost increases by at least L each step, we can go to depth C/L
- Similar memory problems as BFS
  - Iterative lengthening DFS does DFS up to increasing costs

Searching backwards from the goal

- Sometimes can search backwards from the goal
  - Maze puzzles
  - Eights puzzle
  - Reaching location F
  - What about the goal of “having visited all locations”?
- Need to be able to compute predecessors instead of successors
- What’s the point?

Predecessor branching factor can be smaller than successor branching factor

- Stacking blocks:
  - only action is to add something to the stack



In hand: A, B, C

Start state

In hand: nothing

Goal state

We'll see more of this…

Bidirectional search

- Even better: search from both the start and the goal, in parallel!

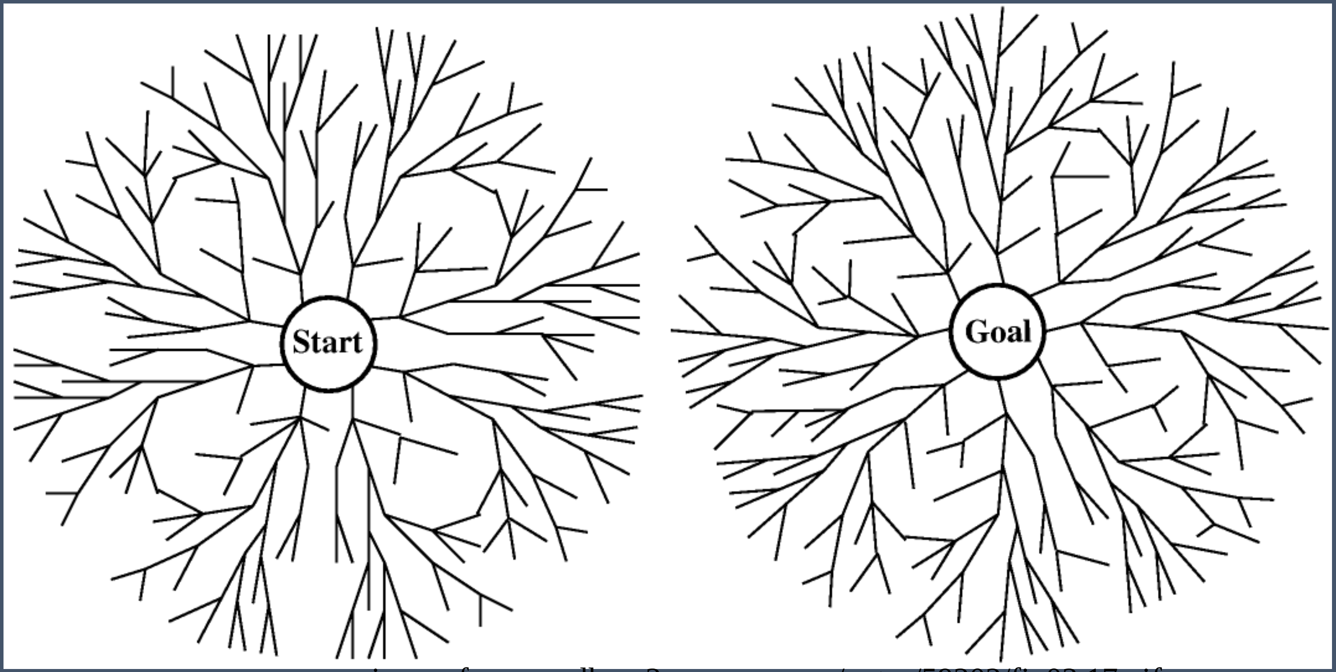


image from cs-alb-pc3.massey.ac.nz/notes/59302/fig03.17.gif

- If the shallowest solution has depth d and branching factor is b on both sides, requires only O($b^{d/2}$) nodes to be explored!

bidirectional search

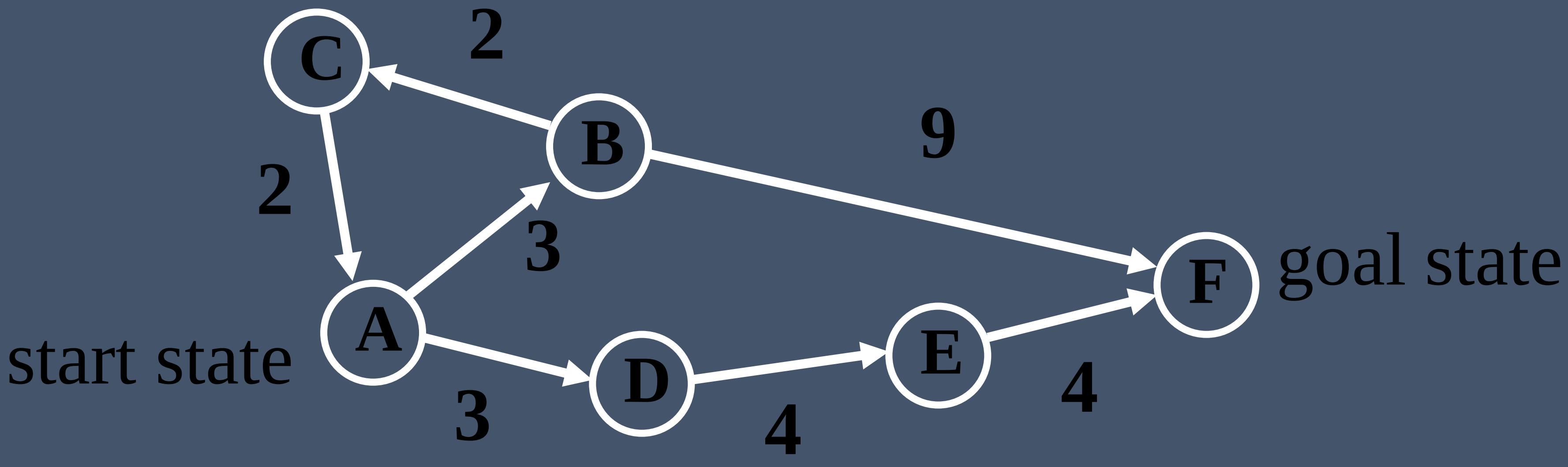
- Need to be able to figure out whether the fringes intersect
  - Need to keep at least one fringe in memory…
- Other than that, can do various kinds of search on either tree, and get the corresponding optimality etc. guarantees
- Not possible (feasible) if backwards search not possible (feasible)
  - Hard to compute predecessors
  - High predecessor branching factor
  - Too many goal states

Informed search

- So far, have assumed that no nongoal state looks better than another
- Unrealistic
  - Even without knowing the road structure, some locations seem closer to the goal than others
  - Some states of the 8s puzzle seem closer to the goal than others
- Makes sense to expand closer-seeming nodes first

Heuristics

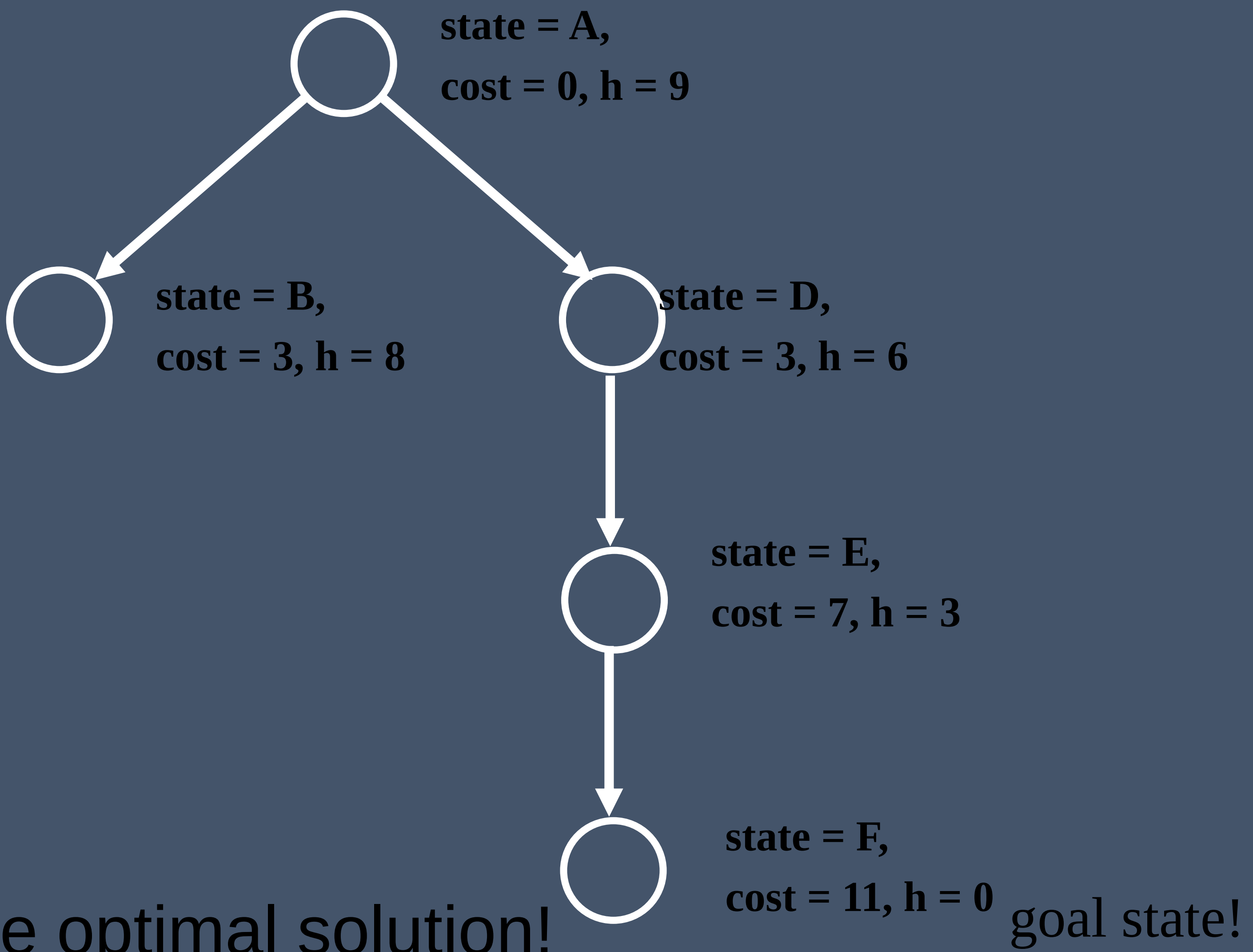
- Key notion: heuristic function h(n) gives an estimate of the distance from n to the goal
  - h(n)=0 for goal nodes
- E.g. straight-line distance for traveling problem



- Say: h(A) = 9, h(B) = 8, h(C) = 9, h(D) = 6, h(E) = 3, h(F) = 0
- We're adding something new to the problem!
- Can use heuristic to decide which nodes to expand first

Greedy best-first search

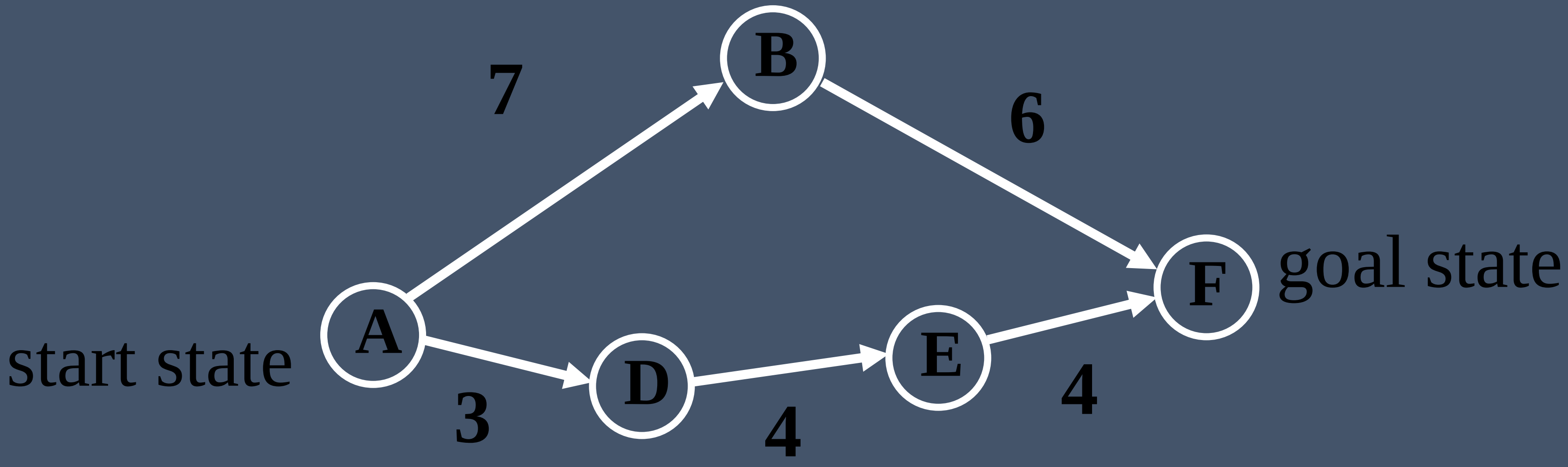
- Greedy best-first search: expand nodes with lowest h values first



- Rapidly finds the optimal solution!
- Does it always?

A*

- Let g(n) be cost incurred already on path to n
- Expand nodes with lowest g(n) + h(n) first



- Say: h(A) = 9, h(B) = 5, h(D) = 6, h(E) = 3, h(F) = 0

- Note: if h=0 everywhere, then just uniform cost search

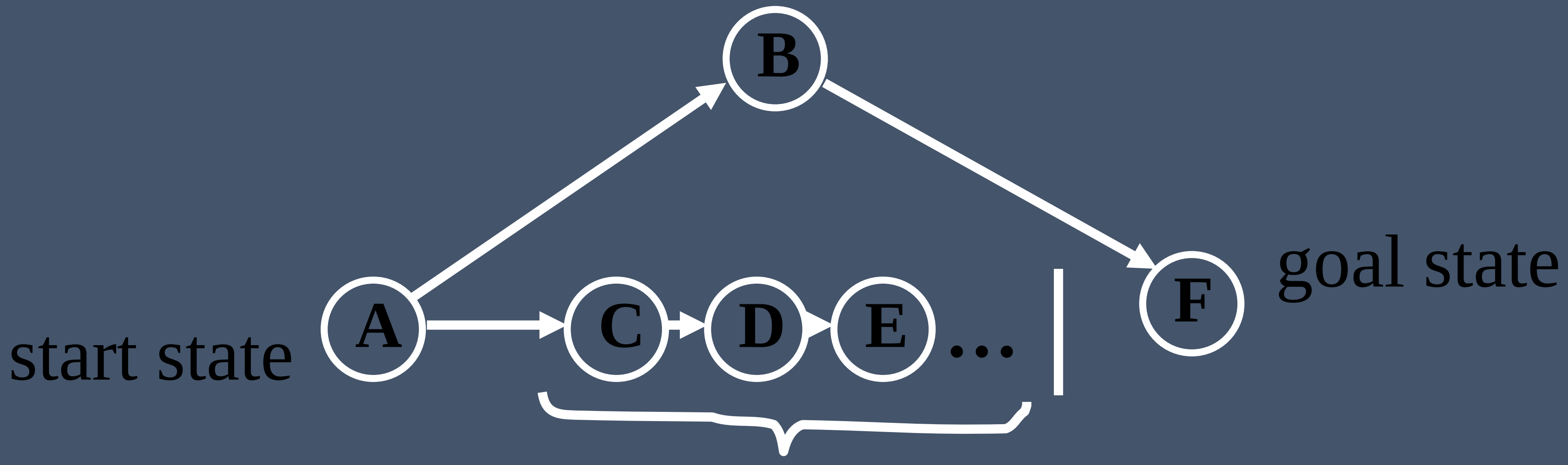
Admissibility

- A heuristic is admissible if it never overestimates the distance to the goal
  - If n is the optimal solution reachable from n’, then $g(n) \geq g(n') + h(n')$
- Straight-line distance is admissible: can’t hope for anything better than a straight road to the goal
- Admissible heuristic means that A* is always optimistic

Optimality of A*

- If the heuristic is admissible, A* is optimal (in the sense that it will never return a suboptimal solution)
- Proof:
  - Suppose a suboptimal solution node n with solution value C > C* is about to be expanded (where C* is optimal)
  - Let n* be an optimal solution node (perhaps not yet discovered)
  - There must be some node n' that is currently in the fringe and on the path to n*
  - We have $g(n) = C > C^* = g(n^*) \geq g(n') + h(n')$
  - But then, n' should be expanded first (contradiction)

A* is not complete (in contrived examples)



infinitely many nodes on a straight path to the goal that doesn’t actually reach the goal

- No optimal search algorithm can succeed on this example (have to keep looking down the path in hope of suddenly finding a solution)

A*

- A* is optimally efficient in the sense that any other optimal algorithm must expand at least the nodes A* expands
- Proof:
  - Besides solution, A* expands exactly the nodes with $g(n)+h(n) < C^*$
    - Assuming it does not expand non-solution nodes with $g(n)+h(n) = C^*$
  - Any other optimal algorithm must expand at least these nodes (since there may be a better solution there)
- Note: This argument assumes that the other algorithm uses the same heuristic h

A*

- Suppose we try to avoid repeated states
- Ideally, the second (or third, …) time that we reach a state the cost is at least as high as the first time
  - Otherwise, have to update everything that came after
- This is guaranteed if the heuristic is consistent: if one step takes us from n to n', then $h(n) \leq h(n')$ + cost of step from n to n'
  - Similar to triangle inequality

Iterative Deepening A*

- One big drawback of A* is the space requirement: similar problems as uniform cost search, BFS
- Limited-cost depth-first A*: some cost cutoff c, any node with g(n)+h(n) > c is not expanded, otherwise DFS
- IDA* gradually increases the cutoff of this
- Can require lots of iterations
  - Trading off space and time…
  - RBFS algorithm reduces wasted effort of IDA*, still linear space requirement
  - SMA* proceeds as A* until memory is full, then starts doing other things

More about heuristics

1		2
4	5	3
7	8	6

- One heuristic: number of misplaced tiles
- Another heuristic: sum of Manhattan distances of tiles to their goal location
  - Manhattan distance = number of moves required if no other tiles are in the way
- Admissible? Which is better?
- Admissible heuristic h_1 dominates admissible heuristic h_2 if $h_1(n) \geq h_2(n)$ for all n
  - Will result in fewer node expansions
- "Best" heuristic of all: solve the remainder of the problem optimally with search
  - Need to worry about computation time of heuristics…

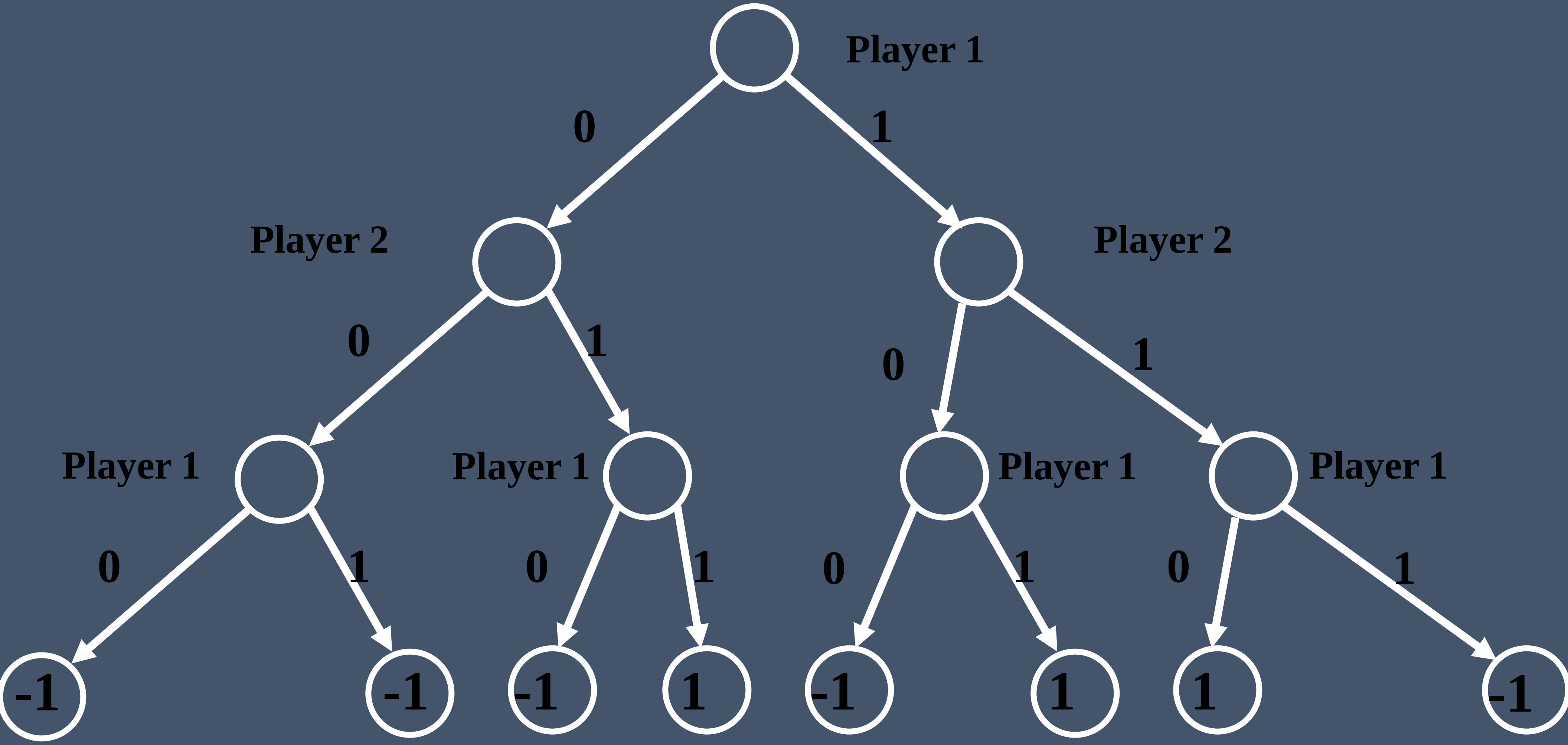
Designing heuristics

- One strategy for designing heuristics: relax the problem (make it easier)
- *"Number of misplaced tiles"* heuristic corresponds to relaxed problem where tiles can jump to any location, even if something else is already there
- *"Sum of Manhattan distances"* corresponds to relaxed problem where multiple tiles can occupy the same spot
- Another relaxed problem: only move 1,2,3,4 into correct locations
- The ideal relaxed problem is
  - easy to solve,
  - not much cheaper to solve than original problem
- Some programs can successfully automatically create heuristics

Game playing

- Rich tradition of creating game-playing programs in AI
- Many similarities to search
- Most of the games studied
  - have two players,
  - are zero-sum: what one player wins, the other loses
  - have perfect information: the entire state of the game is known to both players at all times
- E.g., tic-tac-toe, checkers, chess, Go, backgammon, …
- Will focus on these for now
- Recently more interest in other games
  - Esp. games without perfect information; e.g., poker
    - Need probability theory, game theory for such games

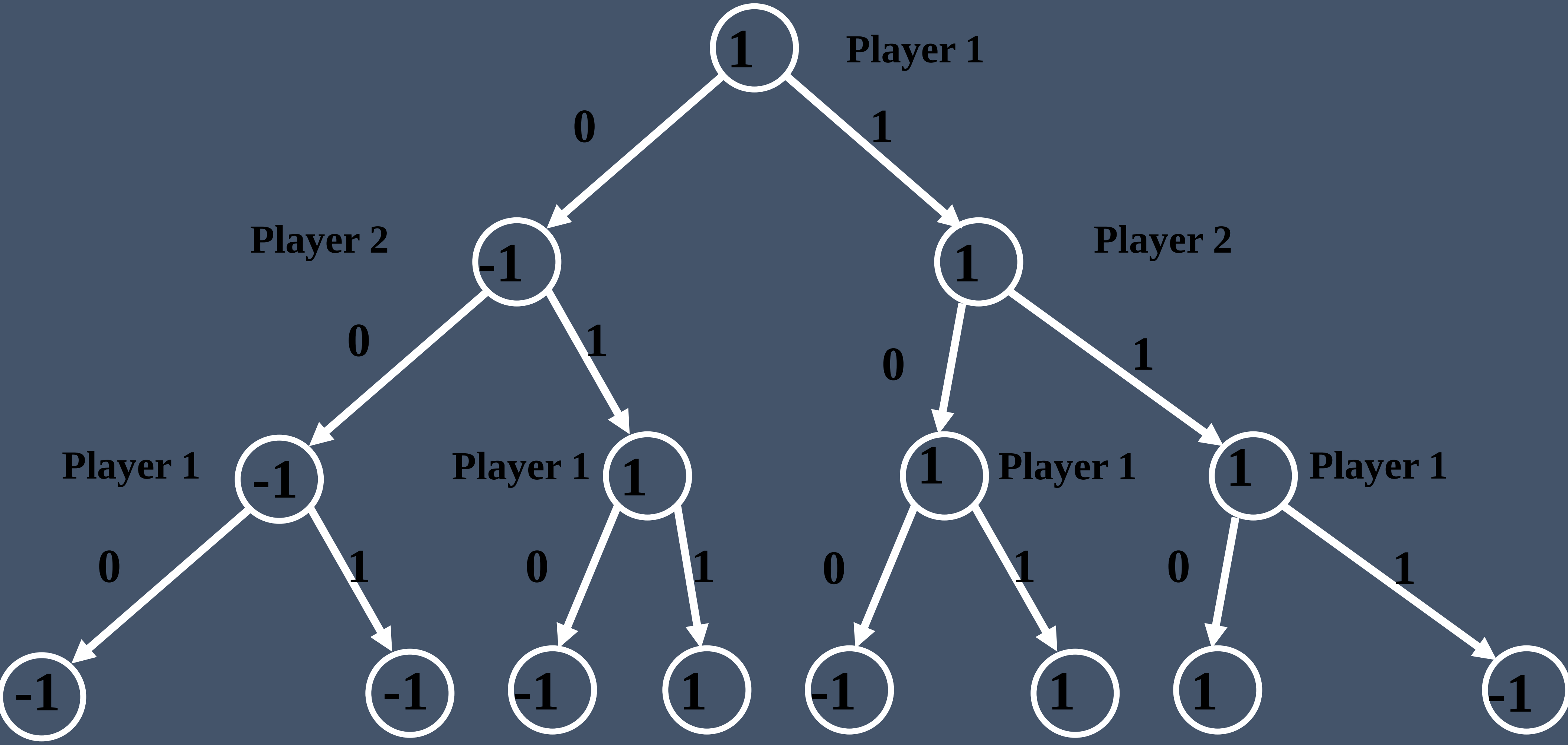
- Player 1 moves, then player 2, finally player 1 again
- Move = 0 or 1
- Player 1 wins if and only if all moves together sum to 2



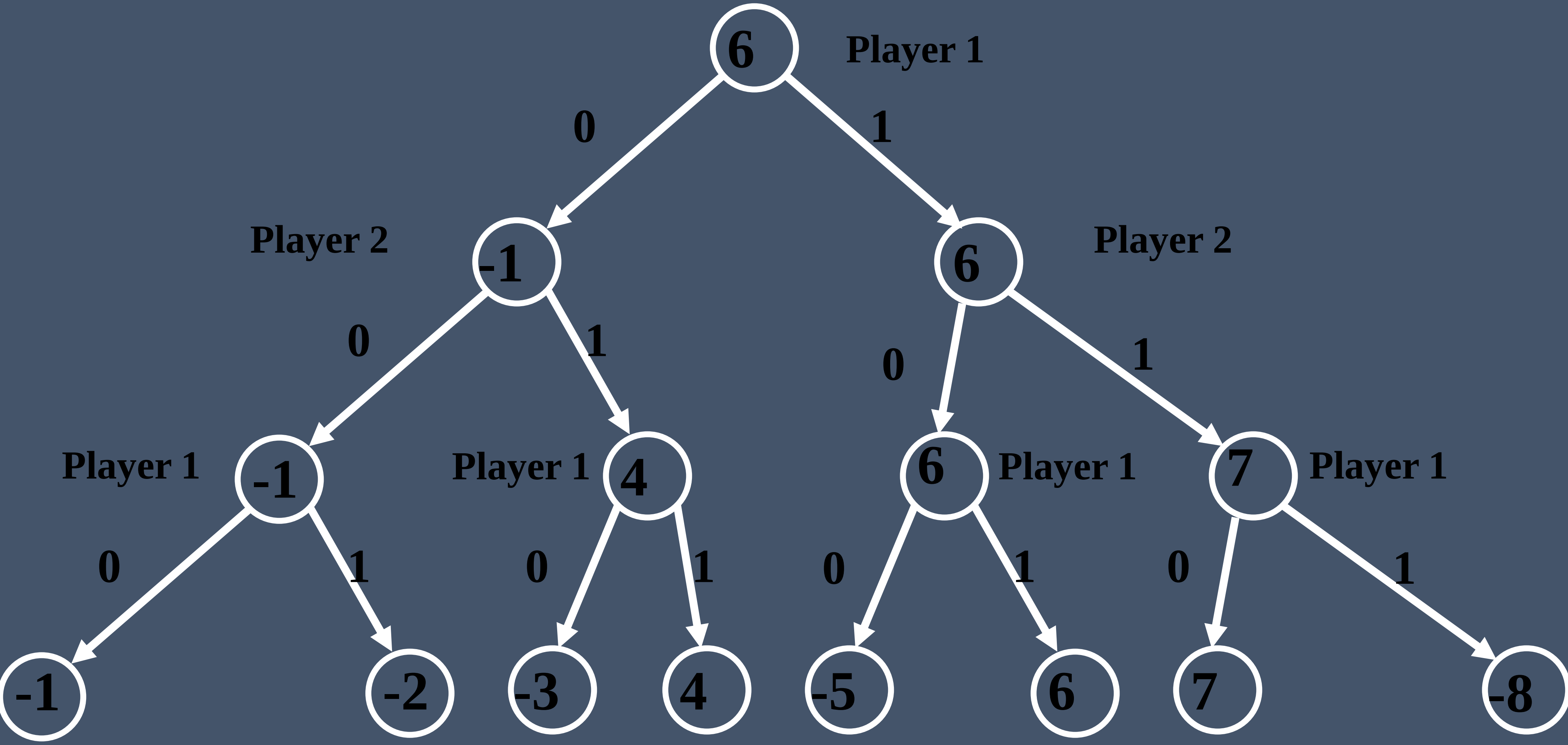
Player 1's utility is in the leaves; player 2's utility is the negative of this

minimax

- From leaves upward, analyze best decision for player at node, give node a value
  - Once we know values, easy to find optimal action (choose best value)



- From leaves upward, analyze best decision for player at node, give node a value

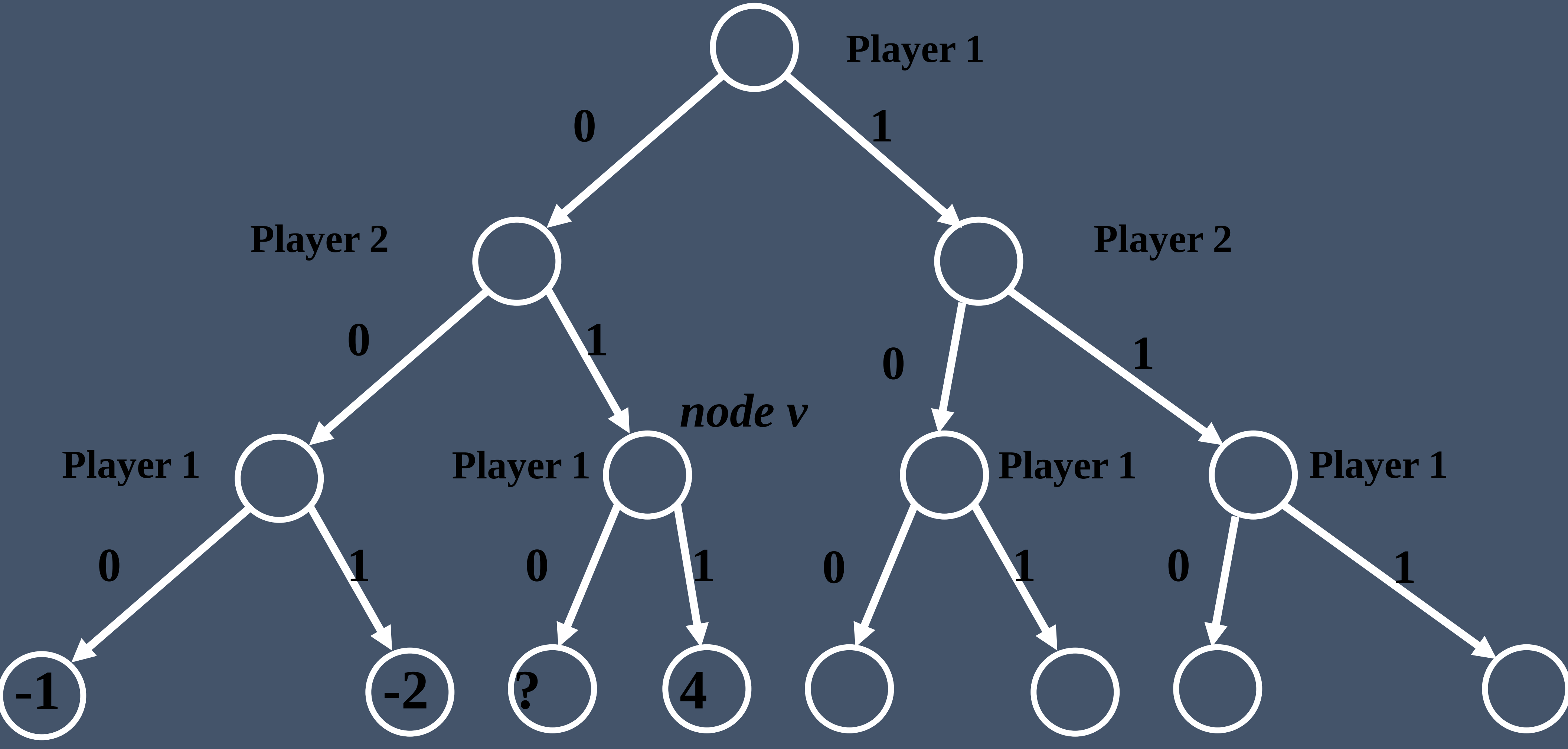


Alpha-beta pruning

- Pruning = cutting off parts of the search tree (because you realize you don’t need to look at them)
  - When we considered A* we also pruned large parts of the search tree
- Maintain alpha = value of the best option for player 1 along the path so far
- Beta = value of the best option for player 2 along the path so far

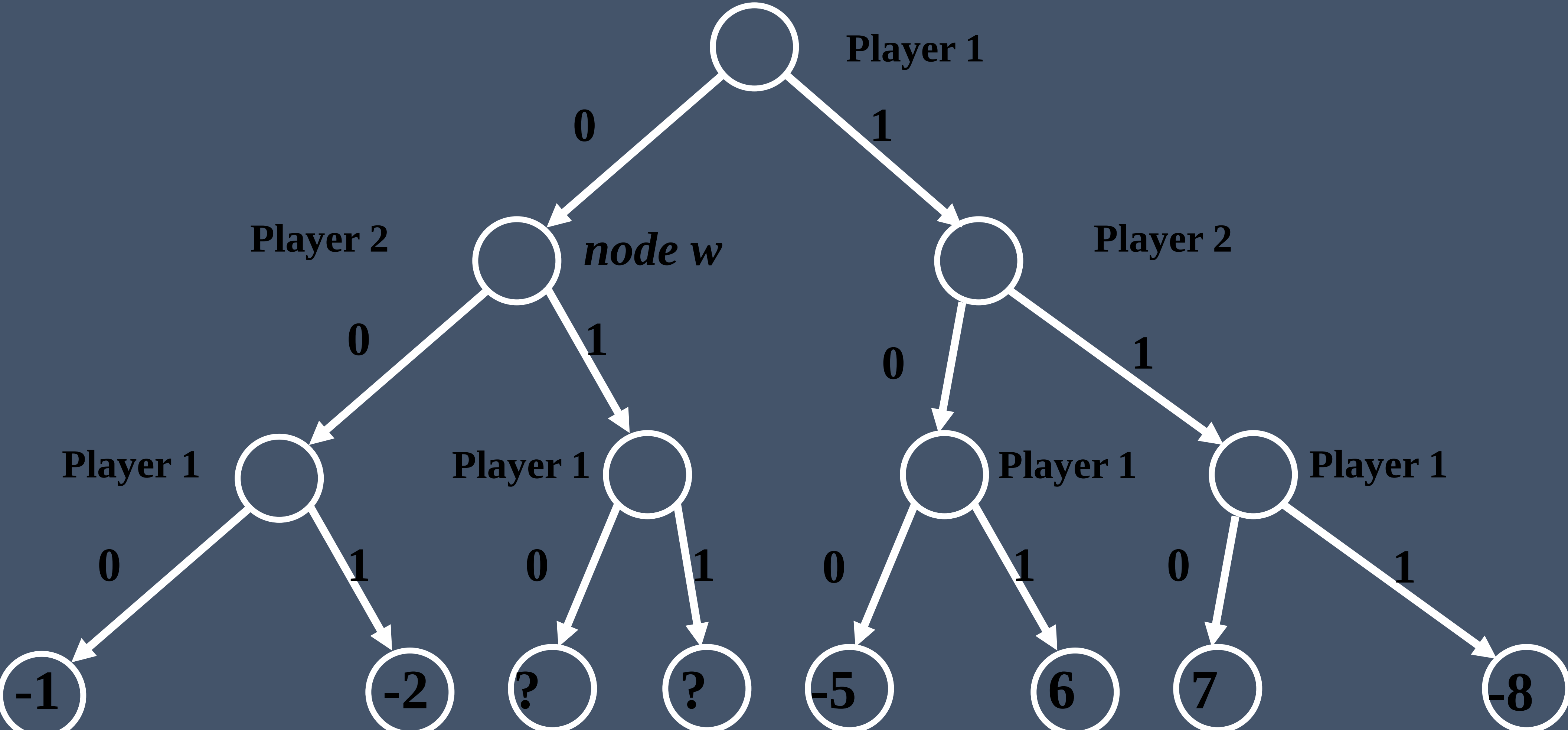
Pruning on beta

- Beta at node v is -1
- We know the value of node v is going to be at least 4, so the -1 route will be preferred
- No need to explore this node further



Pruning on alpha

- Alpha at node w is 6
- We know the value of node w is going to be at most -1, so the 6 route will be preferred
- No need to explore this node further



Benefits of alpha-beta pruning

- Without pruning, need to examine $O(b^m)$ nodes
- With pruning, depends on which nodes we consider first
- If we choose a random successor, need to examine $O(b^{3m/4})$ nodes
- If we manage to choose the best successor first, need to examine $O(b^{m/2})$ nodes
  - Practical heuristics for choosing next successor to consider get quite close to this
- Can effectively look twice as deep!
  - Difference between reasonable and expert play

Repeated states

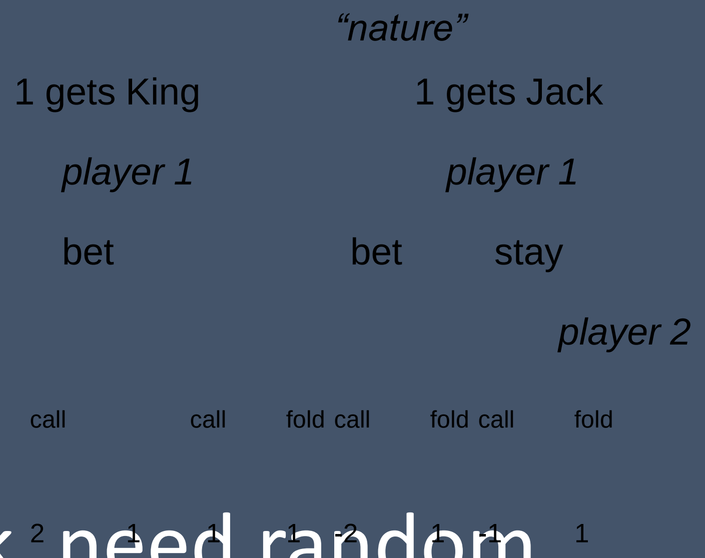
- As in search, multiple sequences of moves may lead to the same state
- Again, can keep track of previously seen states (usually called a transposition table in this context)
  - May not want to keep track of **all** previously seen states…

Using evaluation functions

- Most games are too big to solve even with alpha-beta pruning
- Solution: Only look ahead to limited depth (nonterminal nodes)
- Evaluate nodes at depth cutoff by a heuristic (aka. evaluation function)
- E.g., chess:
  - Material value: queen worth 9 points, rook 5, bishop 3, knight 3, pawn 1
  - Heuristic: difference between players’ material values

Games with imperfect information

- Players cannot necessarily see the whole current state of the game
  - Card games
- Ridiculously simple poker game:
  - Player 1 receives King (winning) or Jack (losing),
  - Player 1 can bet or stay,
  - Player 2 can call or fold
- Dashed lines indicate indistinguishable states
- Backward induction does **not** work, need random strategies for optimality! (more later in course)



Limitations of propositional logic

- So far we studied propositional logic
- Some English statements are hard to model in propositional logic:
- "If your roommate is wet because of rain, your roommate must not be carrying **any** umbrella"
- Pathetic attempt at modeling this:
- RoommateWetBecauseOfRain => (NOT(RoommateCarryingUmbrella0) AND NOT(RoommateCarryingUmbrella1) AND NOT(RoommateCarryingUmbrella2) AND …)

Problems with propositional logic

- No notion of objects
- No notion of relations among objects
- RoommateCarryingUmbrella0 is instructive **to us**, suggesting
    - there is an object we call Roommate,
    - there is an object we call Umbrella0,
    - there is a relationship Carrying between these two objects
- Formally, none of this meaning is there
    - Might as well have replaced RoommateCarryingUmbrella0 by P

Elements of first-order logic

- Objects: can give these names such as Umbrella0, Person0, John, Earth, …
- Relations: Carrying(., .), IsAnUmbrella(.)
  - Carrying(Person0, Umbrella0), IsUmbrella(Umbrella0)
  - Relations with one object = unary relations = properties
- Functions: Roommate(.)
  - Roommate(Person0)
- Equality: Roommate(Person0) = Person1

Reasoning about many objects at once

- Variables: x, y, z, … can refer to multiple objects
- New operators “for all” and “there exists”
    - Universal quantifier and existential quantifier
- for all x: CompletelyWhite(x) => NOT(PartiallyBlack(x))
    - Completely white objects are never partially black
- there exists x: PartiallyWhite(x) AND PartiallyBlack(x)
    - There exists some object in the world that is partially white and partially black

Is this a tautology?

- “Property P implies property Q, or propery P implies property Q (or both)”
- for all x: ((P(x) => Q(x)) OR (Q(x) => P(x)))
- (for all x: (P(x) => Q(x)) OR (for all x: (Q(x) => P(x)))

Relationship between universal and existential

- for all x: a
- is equivalent to
- NOT(there exists x: NOT(a))

Something we can**not** do in first-order logic

- We are **not** allowed to reason in general about relations and functions
- The following would correspond to higher-order logic (which is more powerful):
- “If John is Jack’s roommate, then any property of John is also a property of Jack’s roommate”
- (John=Roommate(Jack)) => for all p: (p(John) => p(Roommate(Jack)))
- “If a property is inherited by children, then for any thing, if that property is true of it, it must also be true for any child of it”
- for all p: (IsInheritedByChildren(p) => (for all x, y: ((IsChildOf(x,y) AND p(y)) => p(x))))

Axioms and theorems

- Axioms: basic facts about the domain, our “initial” knowledge base
- Theorems: statements that are logically derived from axioms

SUBST

- SUBST replaces one or more variables with something else
- For example:
  - SUBST({x/John}, IsHealthy(x) => NOT(HasACold(x))) gives us
  - IsHealthy(John) => NOT(HasACold(John))

Instantiating quantifiers

- From
- for all x: a
- we can obtain
- SUBST({x/g}, a)

- From
- there exists x: a
- we can obtain
- SUBST({x/k}, a)
- where k is a constant that does not appear elsewhere in the knowledge base
- Don’t need original sentence anymore

Instantiating existentials after universals

- for all x: there exists y: IsParentOf(y,x)
- WRONG: for all x: IsParentOf(k, x)
- RIGHT: for all x: IsParentOf(k(x), x)
- Introduces a new function (Skolem function)
- … again, assuming k has not been used previously

modus ponens

- for all x: Loves(John, x)
  - John loves every thing
- for all y: (Loves(y, Jane) => FeelsAppreciatedBy(Jane, y))
  - Jane feels appreciated by every thing that loves her
- Can infer from this:
- FeelsAppreciatedBy(Jane, John)

- Here, we used the substitution {x/Jane, y/John}
  - Note we used different variables for the different sentences
- General UNIFY algorithms for finding a good substitution

Resolution for first-order logic

- for all x: (NOT(Knows(John, x)) OR IsMean(x) OR Loves(John, x))
  - John loves everything he knows, with the possible exception of mean things
- for all y: (Loves(Jane, y) OR Knows(y, Jane))
  - Jane loves everything that does not know her
- What can we unify? What can we conclude?
- Use the substitution: {x/Jane, y/John}
- Get: IsMean(Jane) OR Loves(John, Jane) OR Loves(Jane, John)
- Complete (i.e., if not satisfiable, will find a proof of this), **if** we can remove literals that are duplicates after unification
  - Also need to put everything in canonical form first

inference in first-order logic

- Deciding whether a sentence is entailed is semidecidable: there are algorithms that will eventually produce a proof of any entailed sentence
- It is not decidable: we cannot always conclude that a sentence is not entailed

first-order logic

- You know the following things:
  - You have exactly one other person living in your house, who is wet
  - If a person is wet, it is because of the rain, the sprinklers, or both
  - If a person is wet because of the sprinklers, the sprinklers must be on
  - If a person is wet because of rain, that person must not be carrying any umbrella
  - There is an umbrella that “lives in” your house, which is not in its house
  - An umbrella that is not in its house must be carried by some person who lives in that house
  - You are not carrying any umbrella
- Can you conclude that the sprinklers are on?

Applications

- Some serious novel mathematical results proved
- Verification of hardware and software
    - Prove outputs satisfy required properties for all inputs
- Synthesis of hardware and software
    - Try to prove that there exists a program satisfying such and such properties, in a constructive way
- Also: contributions to planning (up next)

Planning

- We studied how to take actions in the world (search)
- We studied how to represent objects, relations, etc. (logic)
- Now we will combine the two!

State of the world (STRIPS language)

- State of the world = conjunction of positive, ground, function-free literals
- At(Home) AND IsAt(Umbrella, Home) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND HandEmpty AND Dry
- Not OK as part of the state:
  - NOT(At(Home)) (negative)
  - At(x) (not ground)
  - At(Bedroom(Home)) (uses the function Bedroom)
- Any literal not mentioned is assumed false
  - Other languages make different assumptions, e.g., negative literals part of state, unmentioned literals unknown

An action: TakeObject

- TakeObject(location, x)
- Preconditions:
  - HandEmpty
  - CanBeCarried(x)
  - At(location)
  - IsAt(x, location)
- Effects (“NOT something” means that that something should be removed from state):
  - Holding(x)
  - NOT(HandEmpty)
  - NOT(IsAt(x, location))

Another action

- WalkWithUmbrella(location1, location2, umbr)
- Preconditions:
  - At(location1)
  - Holding(umbr)
  - IsUmbrella(umbr)
- Effects:
  - At(location2)
  - NOT(At(location1))

Yet another action

- WalkWithoutUmbrella(location1, location2)
- Preconditions:
  - At(location1)
- Effects:
  - At(location2)
  - NOT(At(location1))
  - NOT(Dry)

A goal and a plan

- Goal: At(Work) AND Dry
- Recall initial state:
  - At(Home) AND IsAt(Umbrella, Home) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND HandEmpty AND Dry
- TakeObject(Home, Umbrella)
  - At(Home) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND Dry AND Holding(Umbrella)
- WalkWithUmbrella(Home, Work, Umbrella)
  - At(Work) AND CanBeCarried(Umbrella) AND IsUmbrella(Umbrella) AND Dry AND Holding(Umbrella)

Some start states

Start1: HasTimeForStudy(You) AND Knows(You,Math) AND Knows(You,Coding) AND Knows(You,Writing)

Start2: HasTimeForStudy(You) AND Creative(You) AND Knows(Advisor,AI) AND Knows(Advisor,Math) AND Knows(Advisor,Coding) AND Knows(Advisor,Writing)

(Good luck with that plan…)

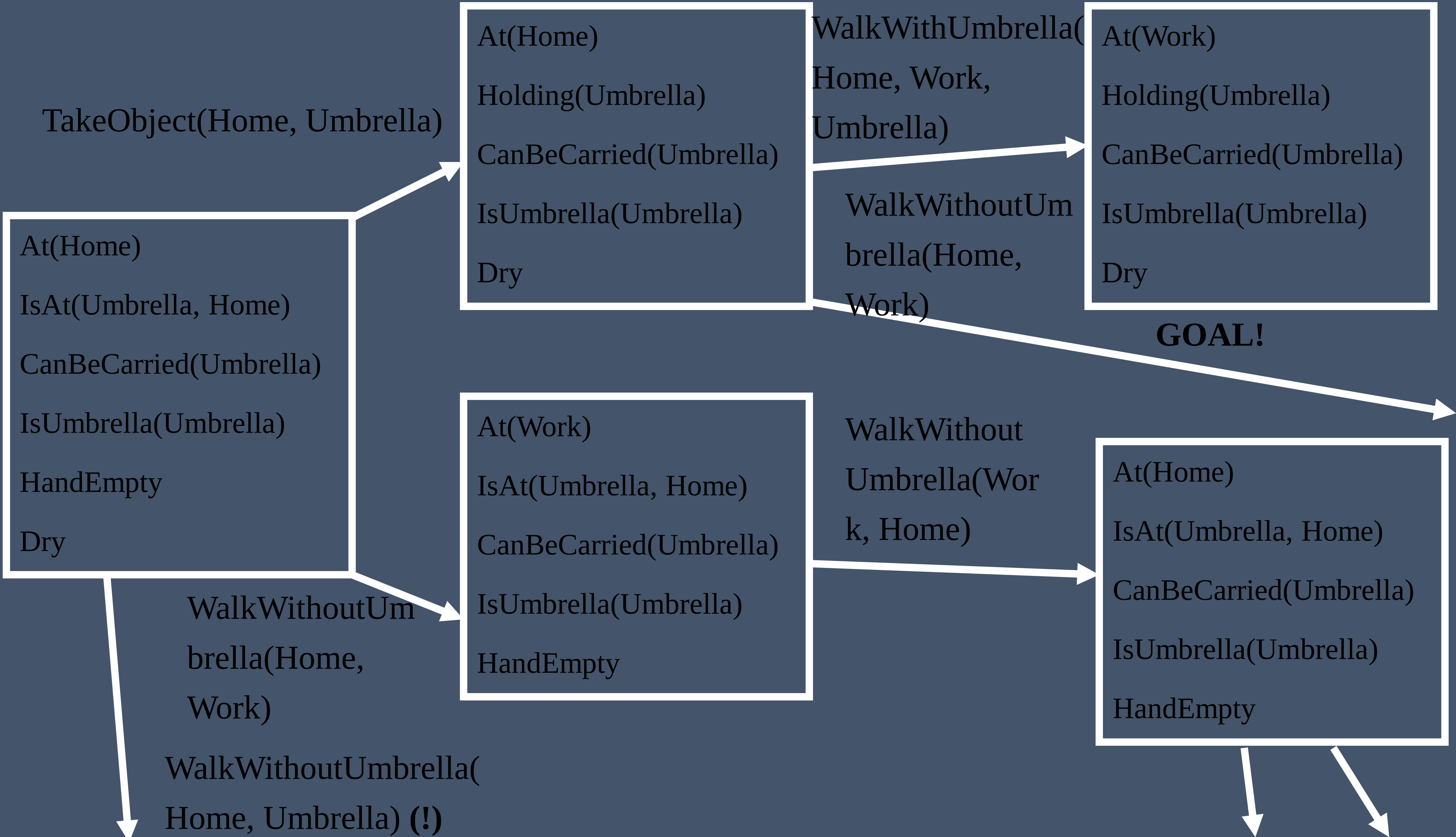
Start3: Knows(You,AI) AND Knows(You,Coding) AND Knows(OfficeMate,Math) AND HasTimeForStudy(OfficeMate) AND Knows(Advisor,AI) AND Knows(Advisor,Writing)

Start4: HasTimeForStudy(You) AND Knows(Advisor,AI) AND Knows(Advisor,Math) AND Knows(Advisor,Coding) AND Knows(Advisor,Writing)

We'll use these as examples…

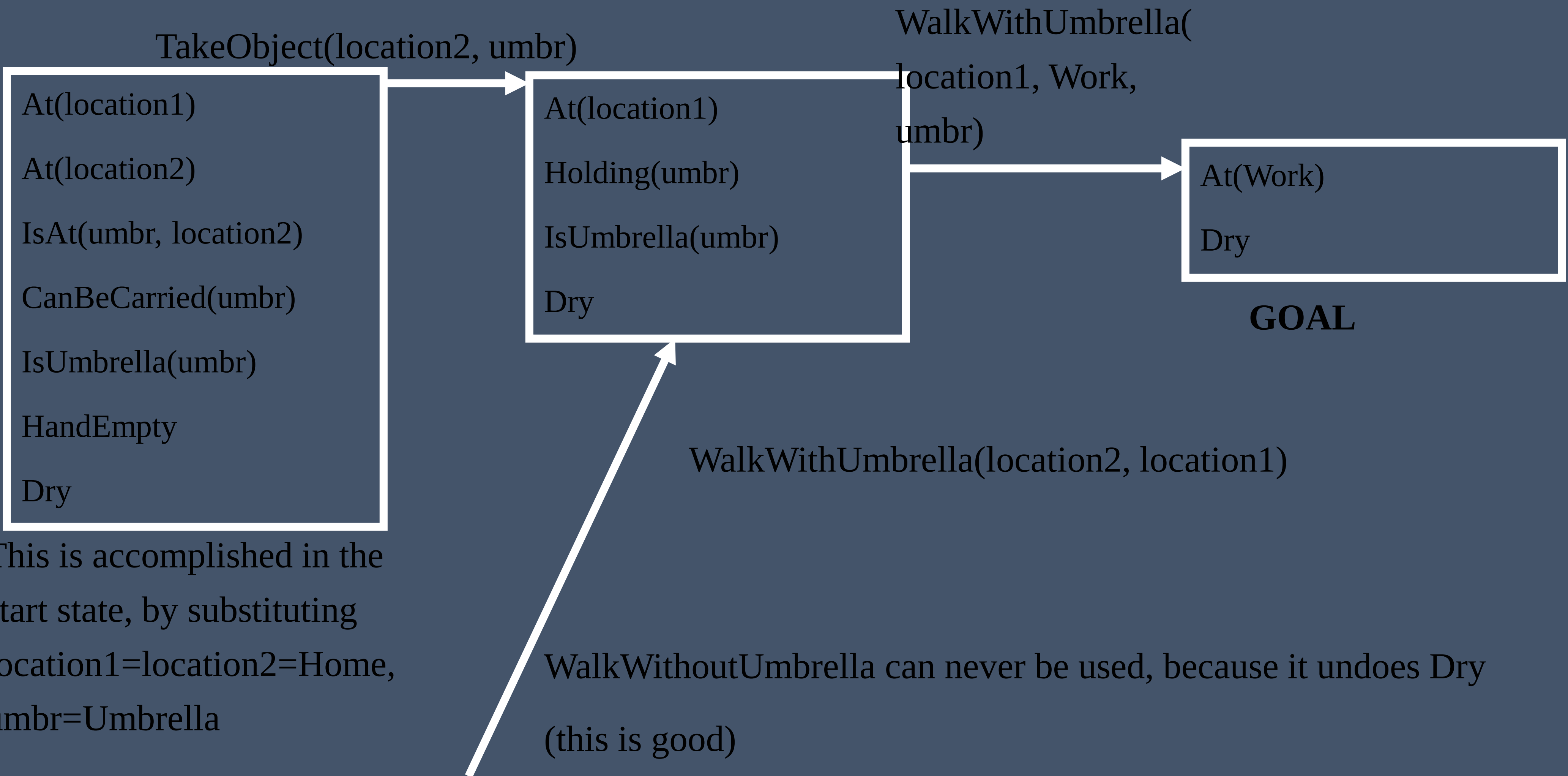
Forward state-space search (progression planning)

- Successors: all states that can be reached with an action whose preconditions are satisfied in current state



Backward state-space search (regression planning)

- Predecessors: for every action that accomplishes one of the literals (and does not undo another literal), remove that literal and add all the preconditions



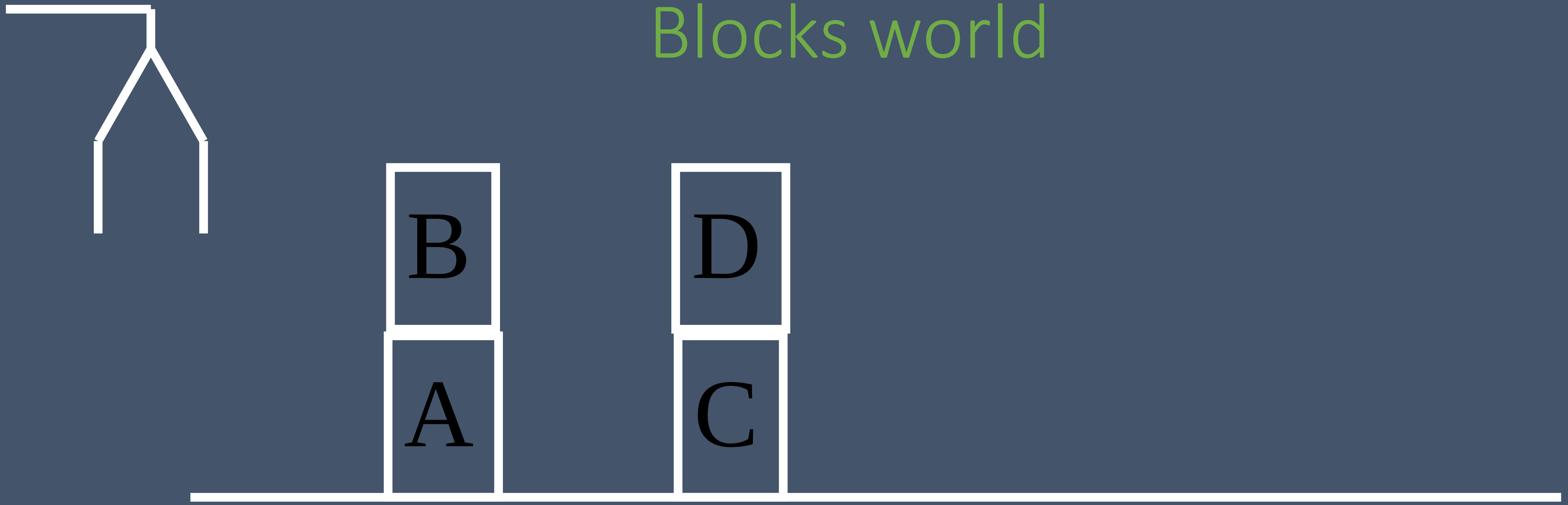
This is accomplished in the start state, by substituting location1=location2=Home, umbr=Umbrella

WalkWithoutUmbrella can never be used, because it undoes Dry (this is good)

Heuristics for state-space search

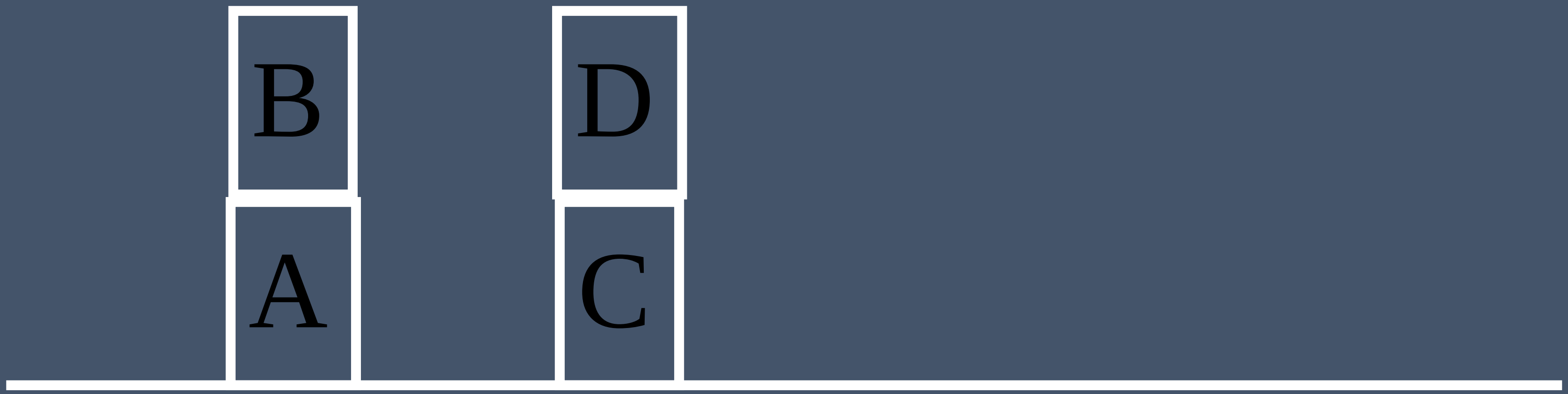
- Cost of a plan: (usually) number of actions
- *Heuristic 1*: plan for each subgoal (literal) separately, sum costs of plans
  - Does this ever underestimate? Overestimate?
- *Heuristic 2*: solve a relaxed planning problem in which actions never delete literals (empty-delete-list heuristic)
  - Does this ever underestimate? Overestimate?
  - Very effective, even though requires solution to (easy) planning problem
- Progression planners with empty-delete-list heuristic perform well

Blocks world



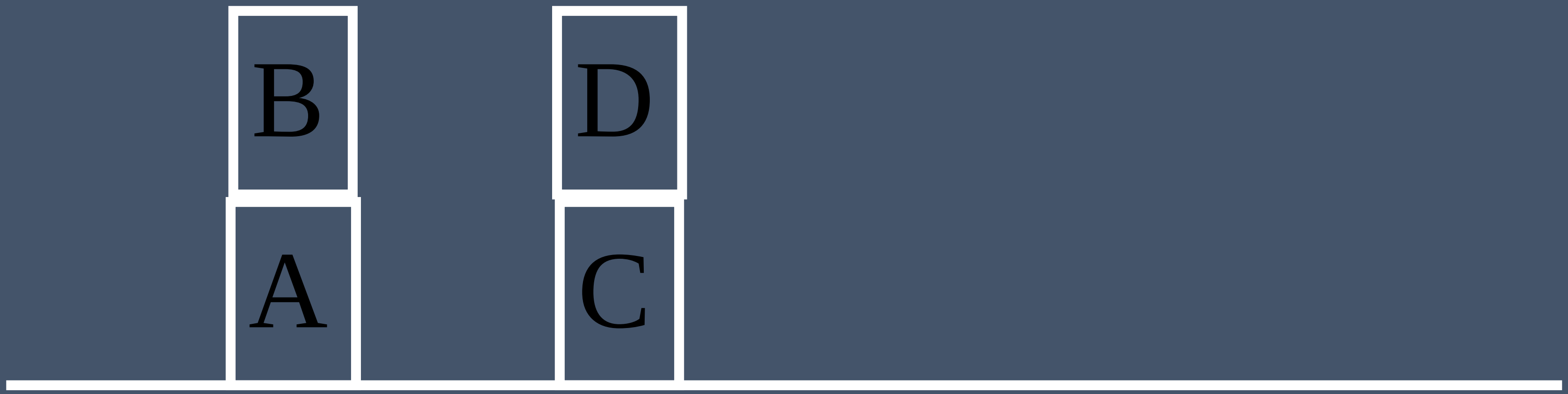
- On(B, A), On(A, Table), On(D, C), On(C, Table), Clear(B), Clear(D)

Blocks world: Move action



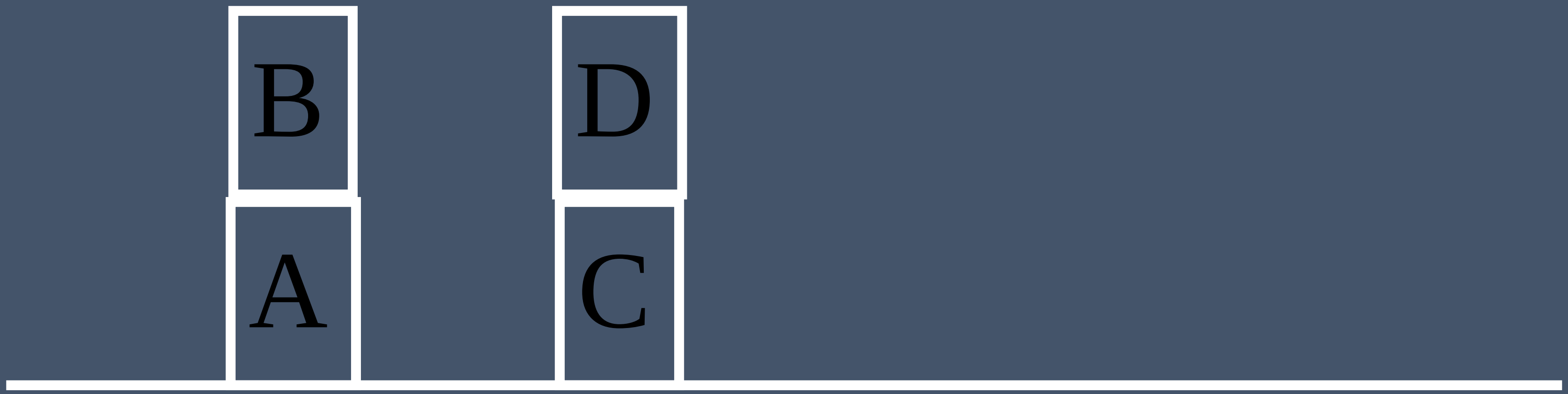
- Move(x,y,z)
- Preconditions:
  - On(x,y), Clear(x), Clear(z)
- Effects:
  - On(x,z), Clear(y), NOT(On(x,y)), NOT(Clear(z))

Blocks world: MoveToTable action



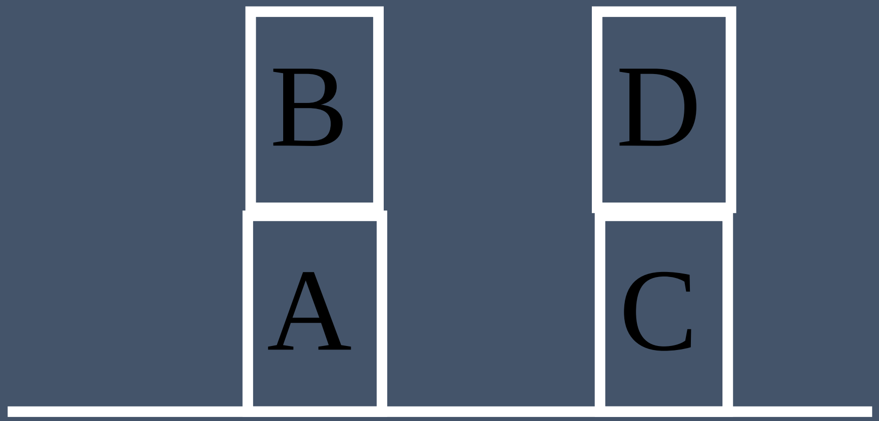
- MoveToTable(x,y)
- Preconditions:
  - On(x,y), Clear(x)
- Effects:
  - On(x,Table), Clear(y), NOT(On(x,y))

Blocks world example

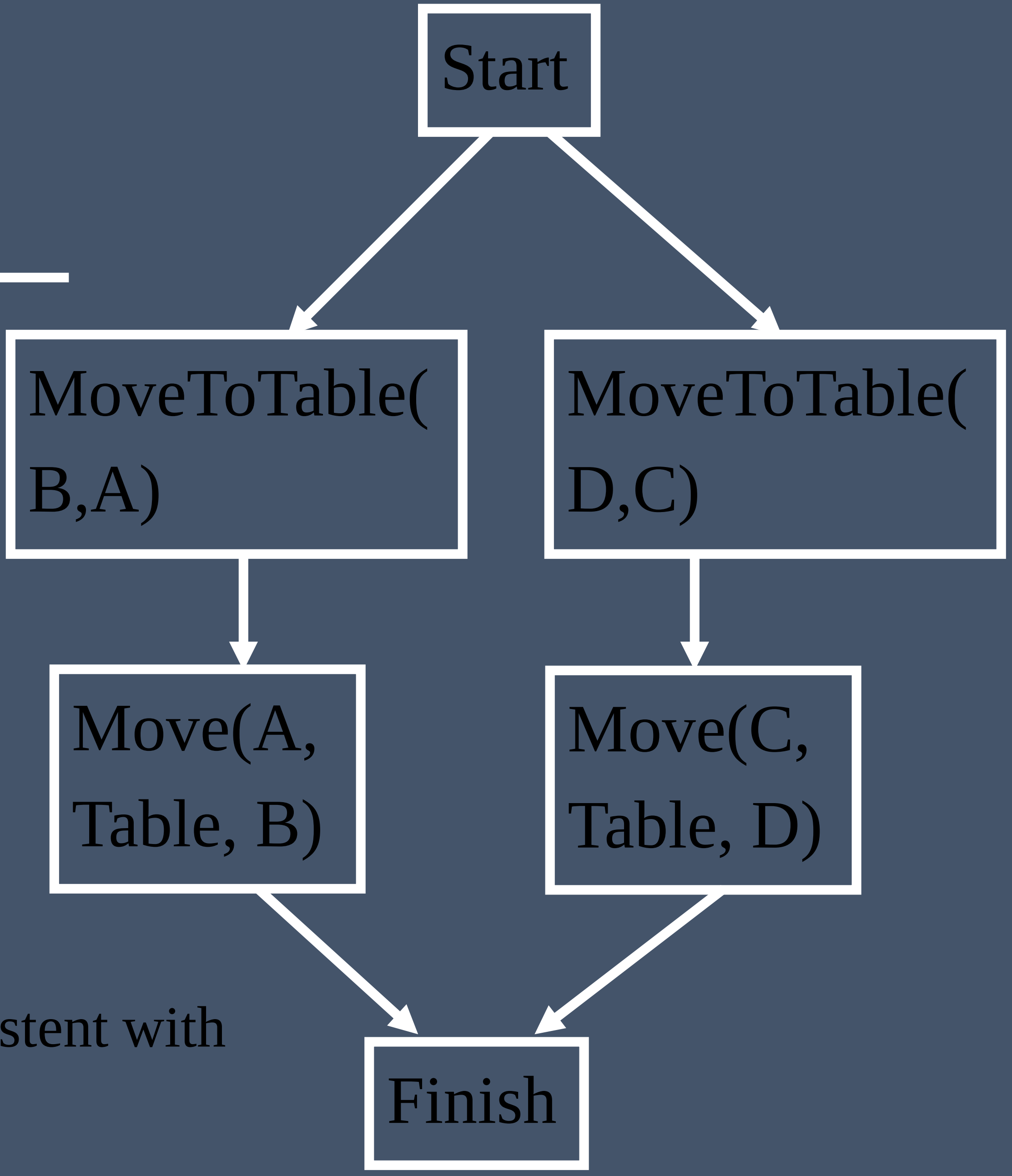


- Goal: On(A,B) AND Clear(A) AND On(C,D) AND Clear(C)
- A plan: MoveToTable(B, A), MoveToTable(D, C), Move(C, Table, D), Move(A, Table, B)
- Really two separate problems

A partial-order plan

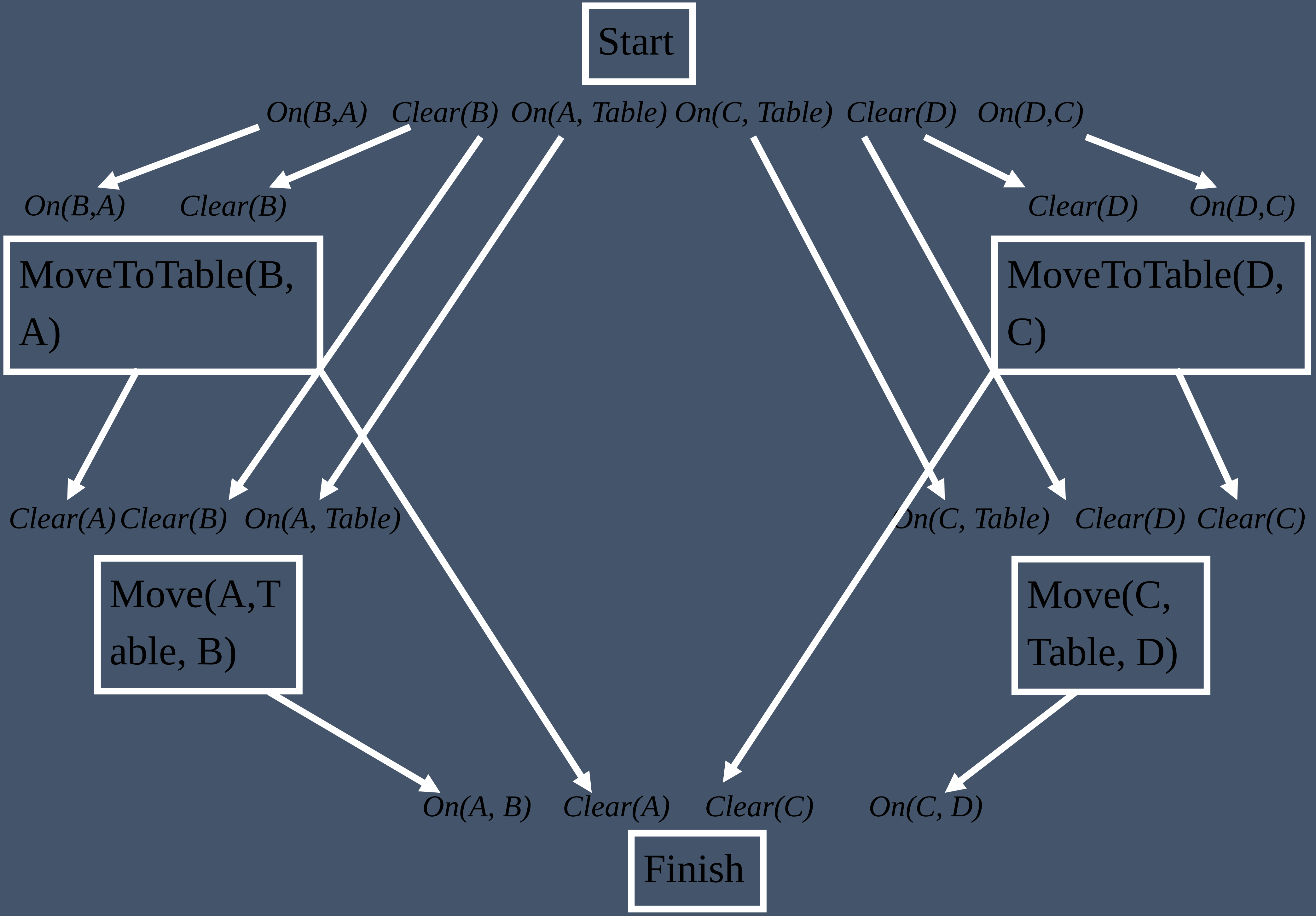


Goal: On(A,B) AND Clear(A) AND On(C,D) AND Clear(C)



Any total order on the actions consistent with this partial order will work

A partial-order plan (with more detail)

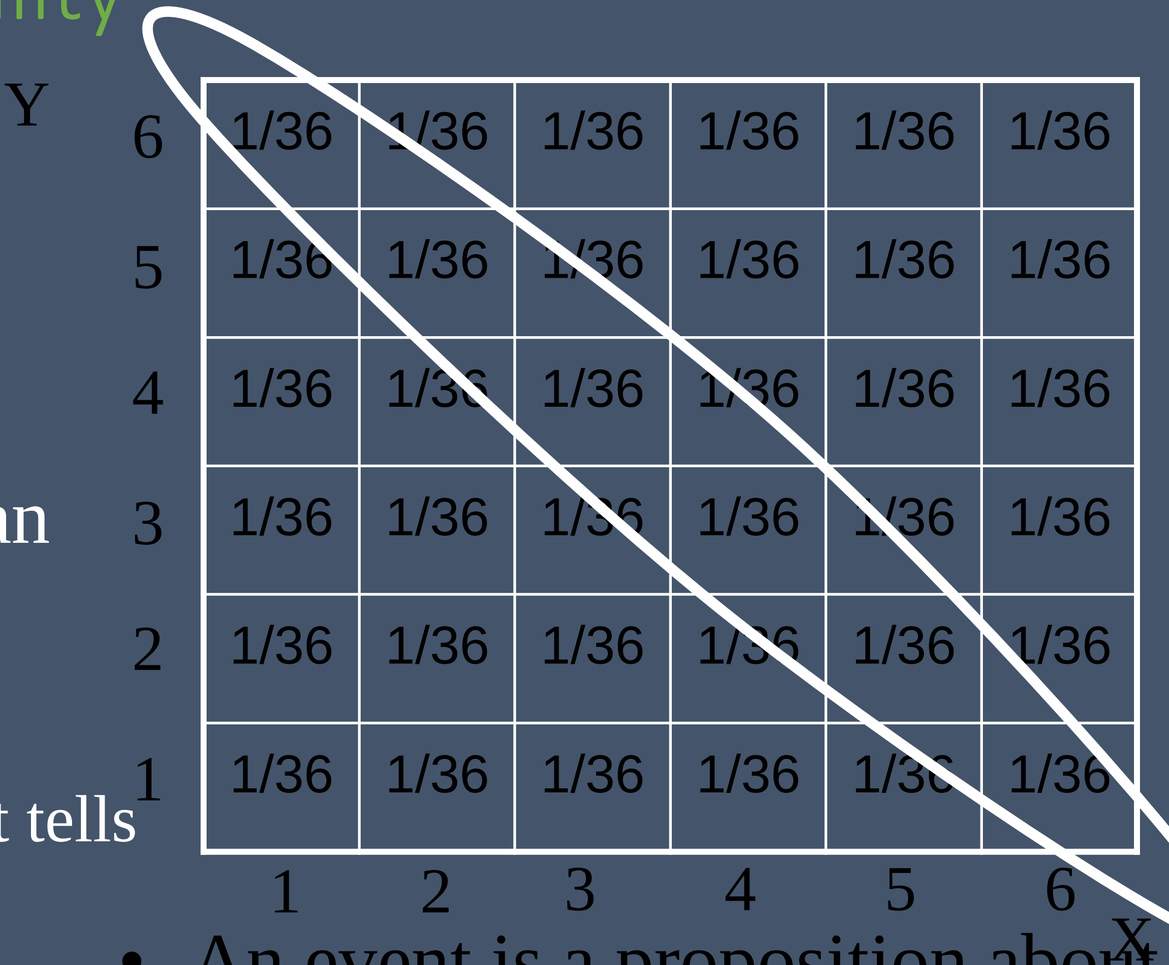


Uncertainty

- So far in course, everything deterministic
- If I walk with my umbrella, I **will not** get wet
- But: there is some chance my umbrella will break!
- Intelligent systems must take possibility of failure into account…
  - May want to have backup umbrella in city that is often windy and rainy
- … but should not be excessively conservative
  - Two umbrellas not worthwhile for city that is usually not windy
- Need quantitative notion of uncertainty

Probability

- Example: roll two dice
- Random variables:
  - X = value of die 1
  - Y = value of die 2
- Outcome is represented by an ordered pair of values (x, y)
  - E.g., (6, 1): X=6, Y=1
  - Atomic event or sample point tells us the **complete** state of the world, i.e., values of **all** random variables
- Exactly one atomic event will happen; each atomic event has a ≥0 probability; sum to 1



Y						
6	1/36	1/36	1/36	1/36	1/36	1/36
5	1/36	1/36	1/36	1/36	1/36	1/36
4	1/36	1/36	1/36	1/36	1/36	1/36
3	1/36	1/36	1/36	1/36	1/36	1/36
2	1/36	1/36	1/36	1/36	1/36	1/36
1	1/36	1/36	1/36	1/36	1/36	1/36
	1	2	3	4	5	6

X

- An event is a proposition about the state (=subset of states)
  - X+Y = 7
- Probability of event = sum of probabilities of atomic events where event is true

Cards and combinatorics

- Draw a hand of 5 cards from a standard deck with 4*13 = 52 cards (4 suits, 13 ranks each)
- Each of the (52 choose 5) hands has same probability 1/(52 choose 5)
- Probability of event = number of hands in that event / (52 choose 5)
- What is the probability that…
  - no two cards have the same rank?
  - you have a flush (all cards the same suit?)
  - you have a straight (5 cards in order of rank, e.g., 8, 9, 10, J, Q)?
  - you have a straight flush?
  - you have a full house (three cards have the same rank and the two other cards have the same rank)?

Facts about probabilities of events

- If events A and B are disjoint, then
  - P(A or B) = P(A) + P(B)
- More generally:
  - P(A or B) = P(A) + P(B) - P(A and B)
- If events A1, …, An are disjoint and exhaustive (one of them must happen) then P(A1) + … + P(An) = 1
  - Special case: for any random variable, $\sum_x$ P(X=x) = 1
- Marginalization: P(X=x) = $\sum_y$ P(X=x and Y=y)

Conditional probability

- We might know something about the world – e.g., "X+Y=6 or X+Y=7" – given this (and **only** this), what is the probability of Y=5?
- Part of the sample space is eliminated; probabilities are renormalized to sum to 1

Y							
6	1/36	1/36	1/36	1/36	1/36	1/36	
5	1/36	1/36	1/36	1/36	1/36	1/36	
4	1/36	1/36	1/36	1/36	1/36	1/36	
3	1/36	1/36	1/36	1/36	1/36	1/36	
2	1/36	1/36	1/36	1/36	1/36	1/36	
1	1/36	1/36	1/36	1/36	1/36	1/36	
	1	2	3	4	5	6	X

Y							
6	1/11	0	0	0	0	0	
5	1/11	1/11	0	0	0	0	
4	0	1/11	1/11	0	0	0	
3	0	0	1/11	1/11	0	0	
2	0	0	0	1/11	1/11	0	
1	0	0	0	0	1/11	1/11	
	1	2	3	4	5	6	X

- P(Y=5 | (X+Y=6) or (X+Y=7)) = 2/11

Facts about conditional probability

- $P(A \mid B) = P(A \text{ and } B) / P(B)$

- $P(A \mid B)P(B) = P(A \text{ and } B)$
- $P(A \mid B) = P(B \mid A)P(A)/P(B)$
  – Bayes’ rule

Conditional probability and cards

- Given that your first two cards are Queens, what is the probability that you will get at least 3 Queens?
- Given that you have at least two Queens (not necessarily the first two), what is the probability that you have at least three Queens?
- Given that you have at least two Queens, what is the probability that you have three Kings?

How can we scale this?

- In principle, we now have a complete approach for reasoning under uncertainty:
  - Specify probability for every atomic event,
  - Can compute probabilities of events simply by summing probabilities of atomic events,
  - Conditional probabilities are specified in terms of probabilities of events: P(A | B) = P(A and B) / P(B)
- If we have n variables that can each take k values, how many atomic events are there?

Independence

- Some variables have nothing to do with each other
- Dice: if X=6, it tells us nothing about Y
- $P(Y=y \mid X=x) = P(Y=y)$
- So: $P(X=x \text{ and } Y=y) = P(Y=y \mid X=x)P(X=x) = P(Y=y)P(X=x)$
    - Usually just write $P(X, Y) = P(X)P(Y)$
    - Only need to specify 6+6=12 values instead of 6*6=36 values
    - Independence among 3 variables: $P(X,Y,Z)=P(X)P(Y)P(Z)$, etc.
- Are the events “you get a flush” and “you get a straight” independent?

An example without cards or dice

	Rain in Beaufort	Sun in Beaufort
Rain in Durham	.2	.1
Sun in Durham	.2	.5

(disclaimer: no idea if these numbers are realistic)

- What is the probability of
  - Rain in Beaufort? Rain in Durham?
  - Rain in Beaufort, given rain in Durham?
  - Rain in Durham, given rain in Beaufort?
- Rain in Beaufort and rain in Durham are correlated

Conditional independence

- Intuition:
  - the only reason that X told us something about Y,
  - is that X told us something about Z,
  - and Z tells us something about Y
- If we already know Z, then X tells us nothing about Y
- $P(Y \mid Z, X) = P(Y \mid Z)$ or
- $P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z)$
- “X and Y are conditionally independent given Z”

Specifying probability distributions

- Specifying a probability for every atomic event is impractical
- We have already seen it can be easier to specify probability distributions by using (conditional) independence
- Bayesian networks allow us
  - to specify any distribution,
  - to specify such distributions concisely if there is (conditional) independence, in a natural way

A general approach to specifying probability distributions

- Say the variables are $X_1, \ldots, X_n$
- $P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\ldots P(X_n|X_1, \ldots, X_{n-1})$
- Can specify every component
- If every variable can take k values,
- $P(X_i|X_1, \ldots, X_{i-1})$ requires $(k-1)k^{i-1}$ values
- $\Sigma_{i=\{1,..,n\}}(k-1)k^{i-1} = \Sigma_{i=\{1,..,n\}}k^i-k^{i-1} = k^n - 1$
- Same as specifying probabilities of all atomic events – of course, because we can specify any distribution!

Conditional independence to the rescue!

- Problem: $P(X_i | X_1, \ldots, X_{i-1})$ requires us to specify too many values
- Suppose $X_1, \ldots, X_{i-1}$ partition into two subsets, *S* and *T*, so that X_i is conditionally independent from *T* given *S*
- $P(X_i | X_1, \ldots, X_{i-1}) = P(X_i | S, T) = P(X_i | S)$
- Requires only $(k-1)k^{|S|}$ values instead of $(k-1)k^{i-1}$ values

References

- Textbook: *Artificial Intelligence: A Modern Approach*, Stuart Russell and Peter Norvig.