

Path Planning as Search



Paul Robertson

16.410 16.413

Session 7

Slides adapted from:

Brian C. Williams

6.034 Tomas Lozano Perez,

Winston, and Russell and Norvig AIMA

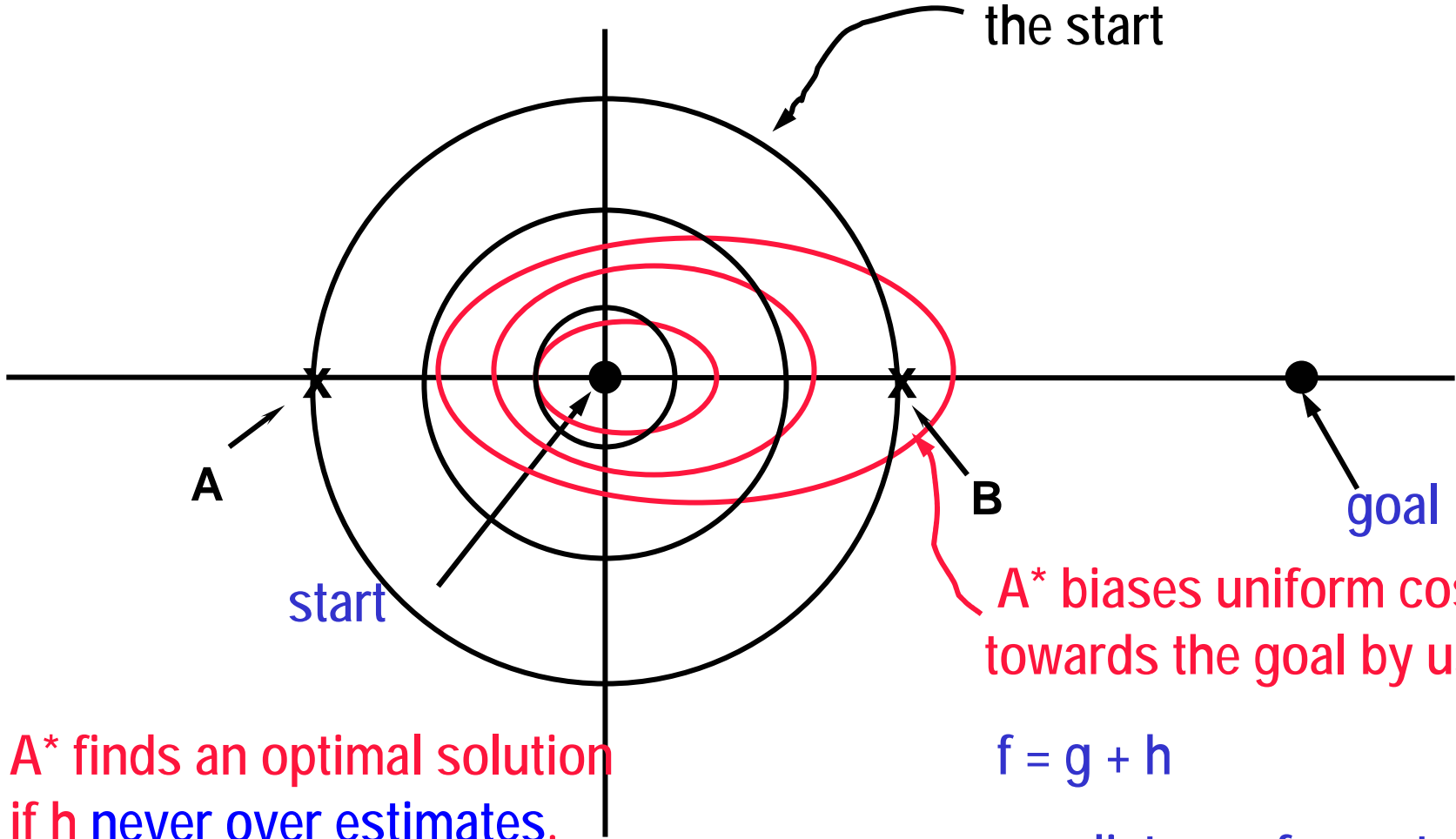
Assignment

- **Remember:**
Online problem set #3 due Session 8
Hand in written part in class.
- **Reading:**
 - Adversarial Search: AIMA Ch. 6
 - From before: Path planning: AIMA Ch. 25.4

Roadmaps are an effective state space abstraction



Uniform cost search spreads evenly from the start



A* finds an optimal solution if h never over estimates.

Then h is called "admissible"

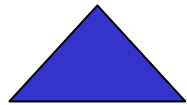
A* biases uniform cost towards the goal by using h

$$f = g + h$$

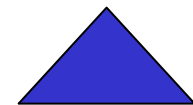
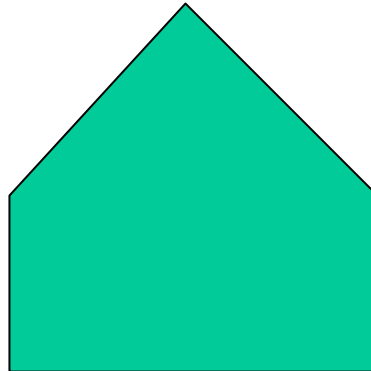
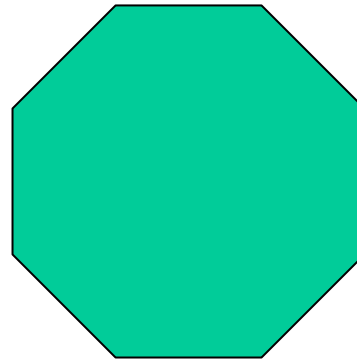
g = distance from start

h = estimated distance to goal.

Path Planning through Obstacles

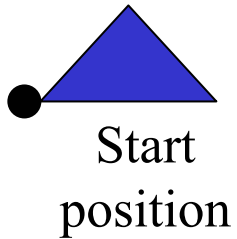


Start
position

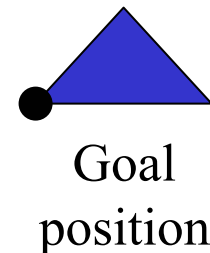
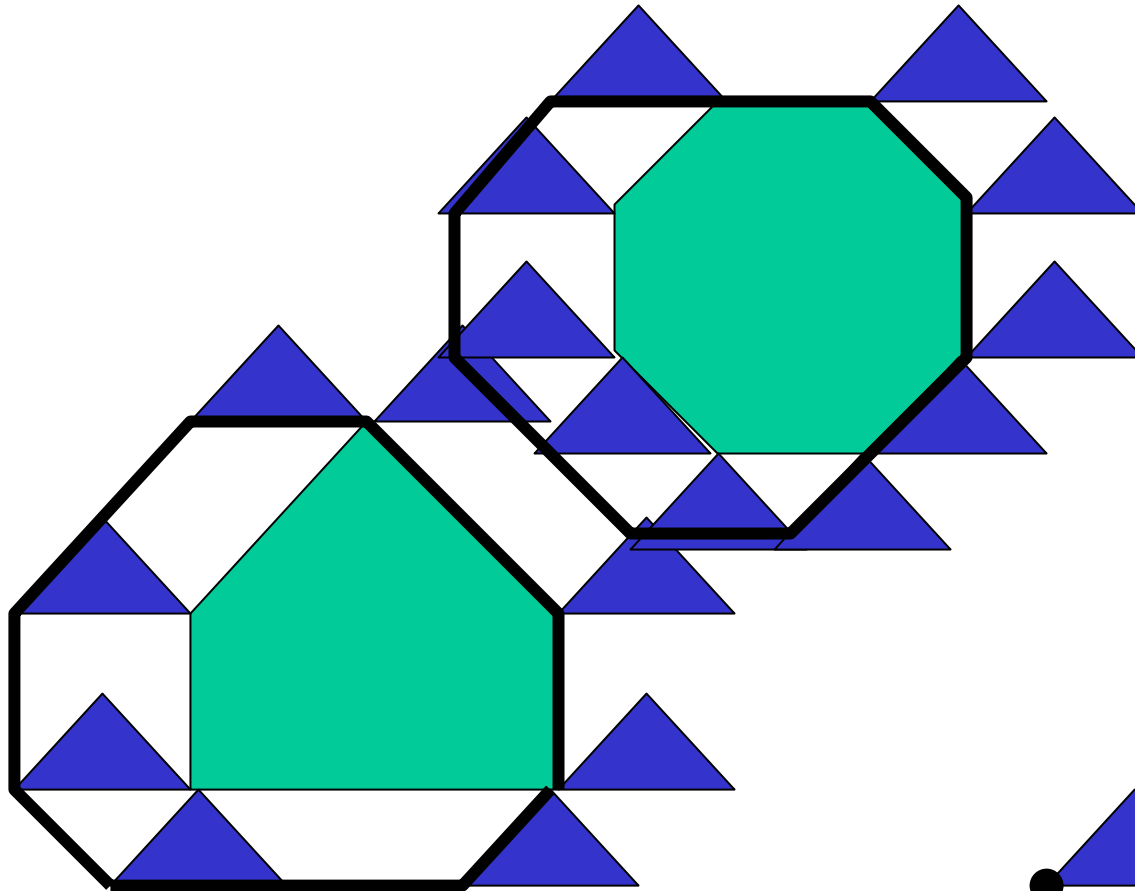


Goal
position

1. Create Configuration Space



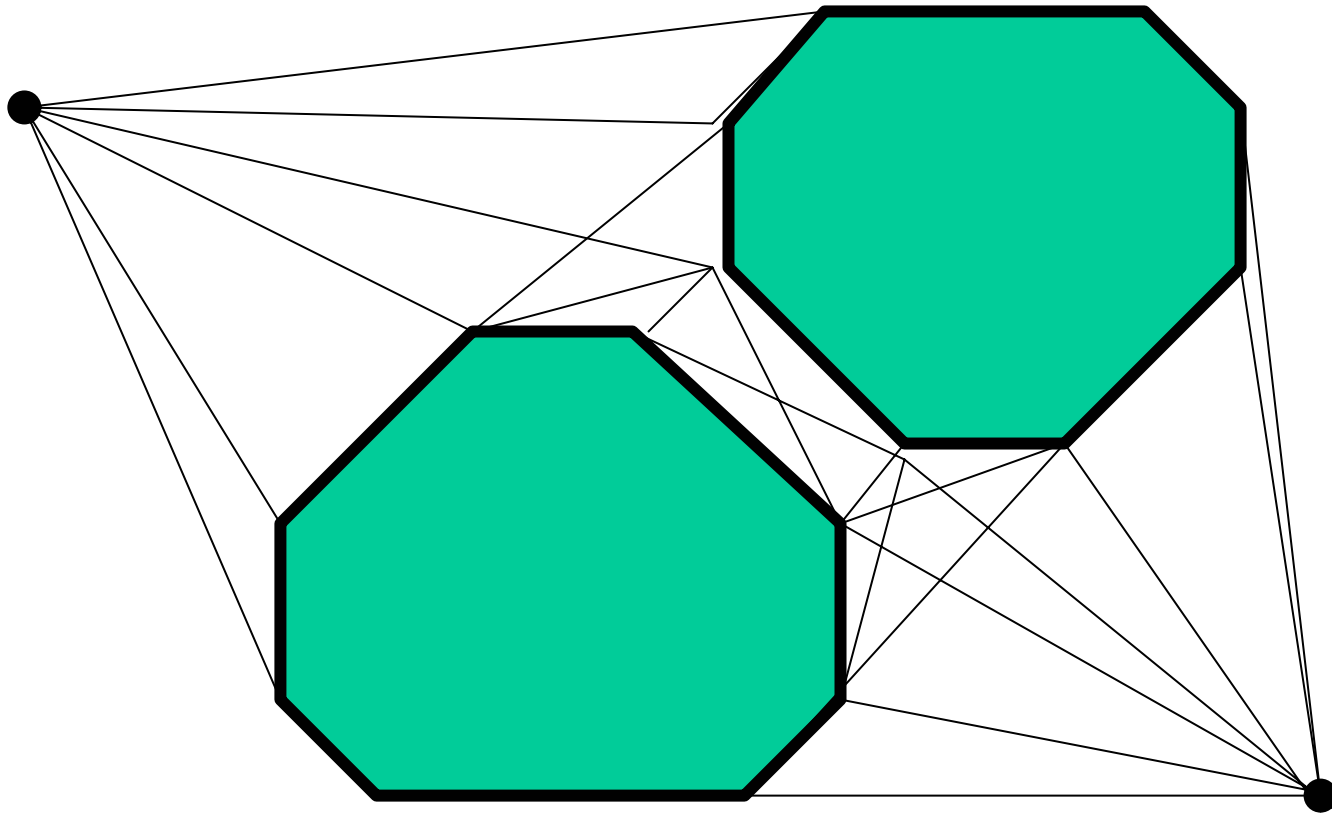
Vehicle translates,
but no rotation



Idea: Transform to equivalent
Problem of navigating a point.

2. Map From Continuous Problem to Graph Search: Create Visibility Graph

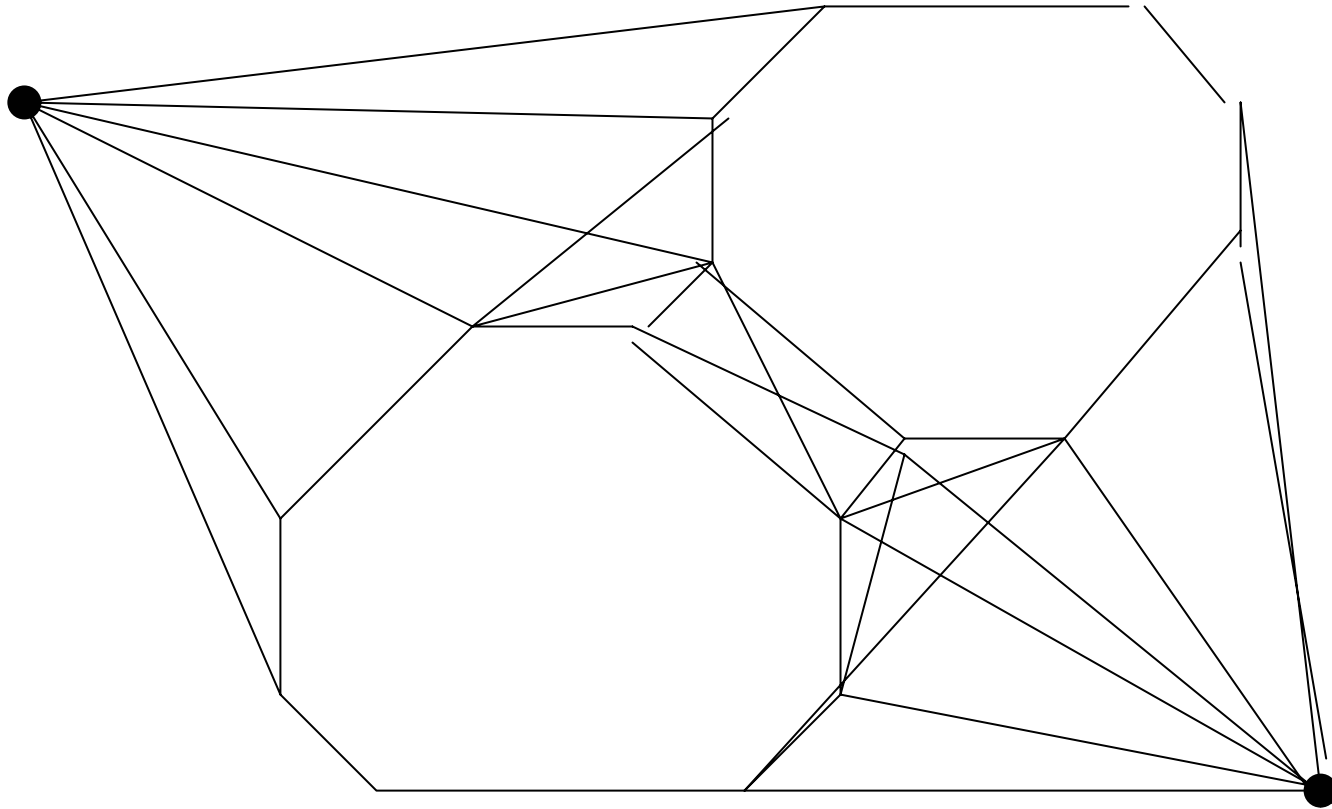
Start
position



Goal
position

2. Map From Continuous Problem to Graph Search: Create Visibility Graph

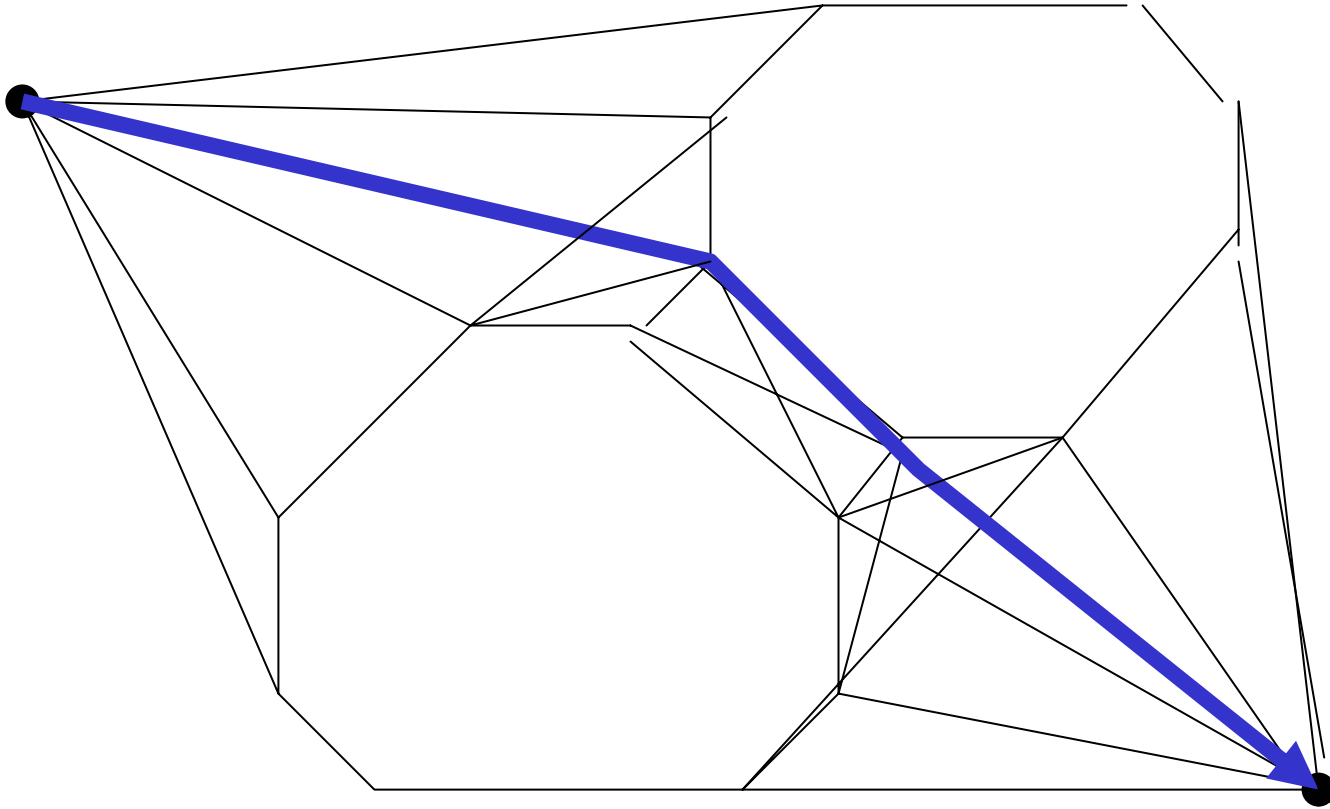
Start
position



Goal
position

3. Find Shortest Path

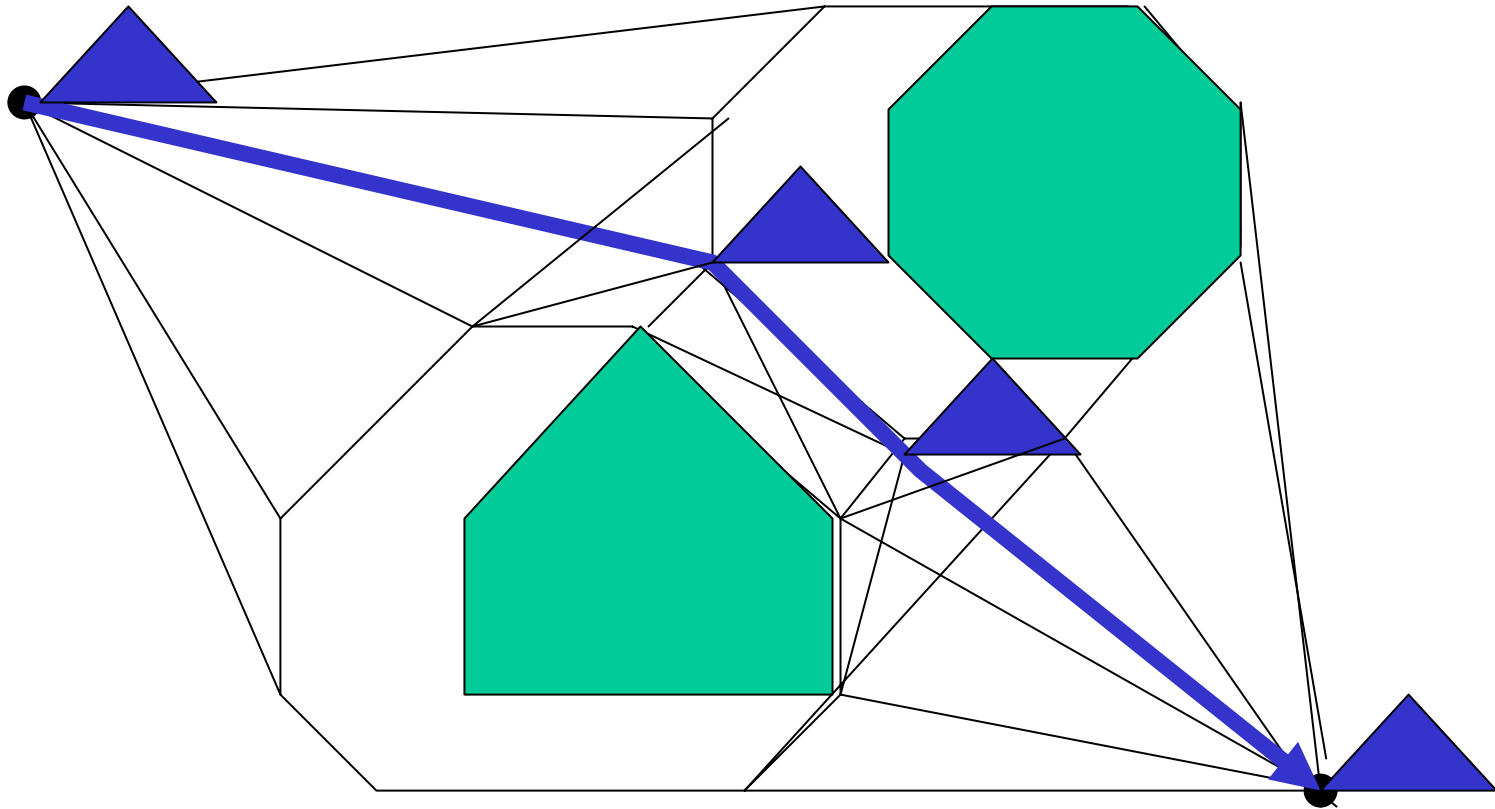
Start
position



Goal
position

Resulting Solution

Start
position

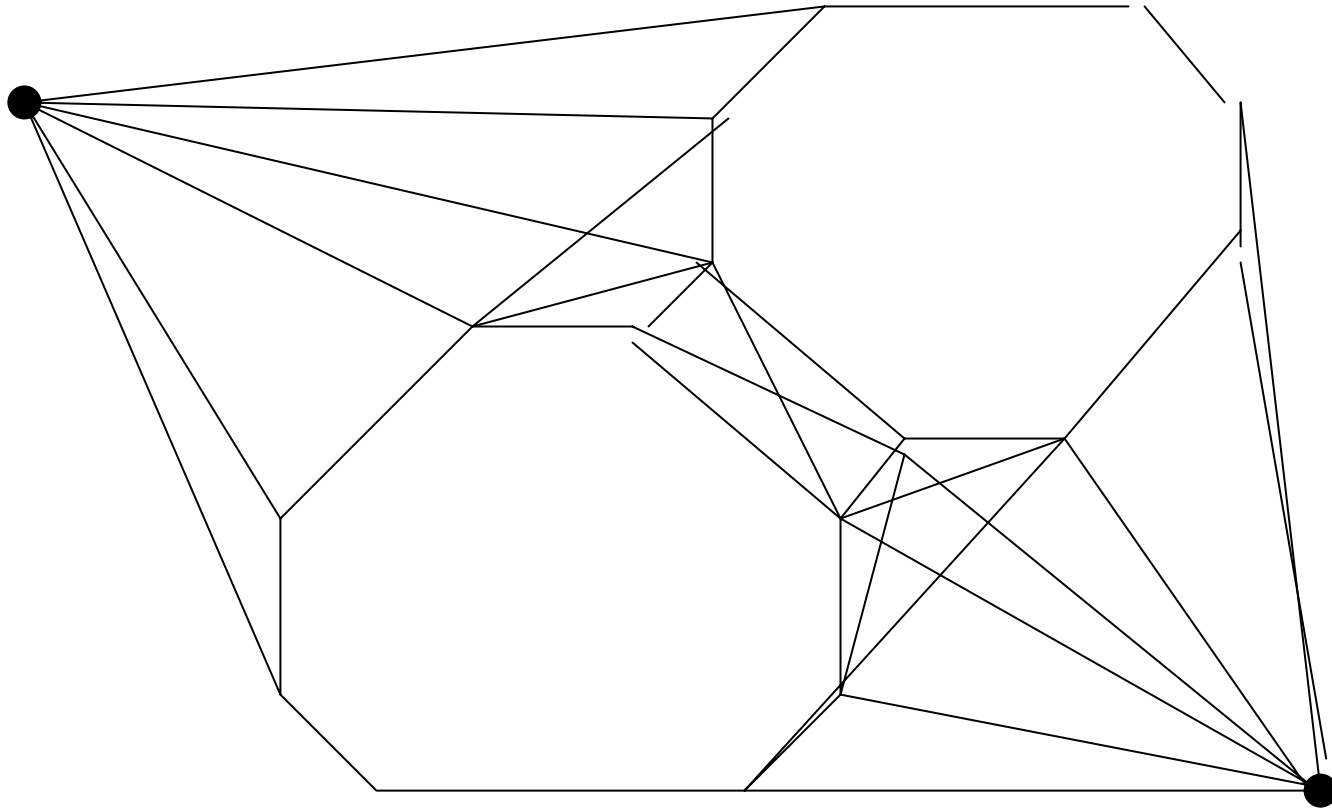


Goal
position

A Visibility Graph is a Kind of Roadmap

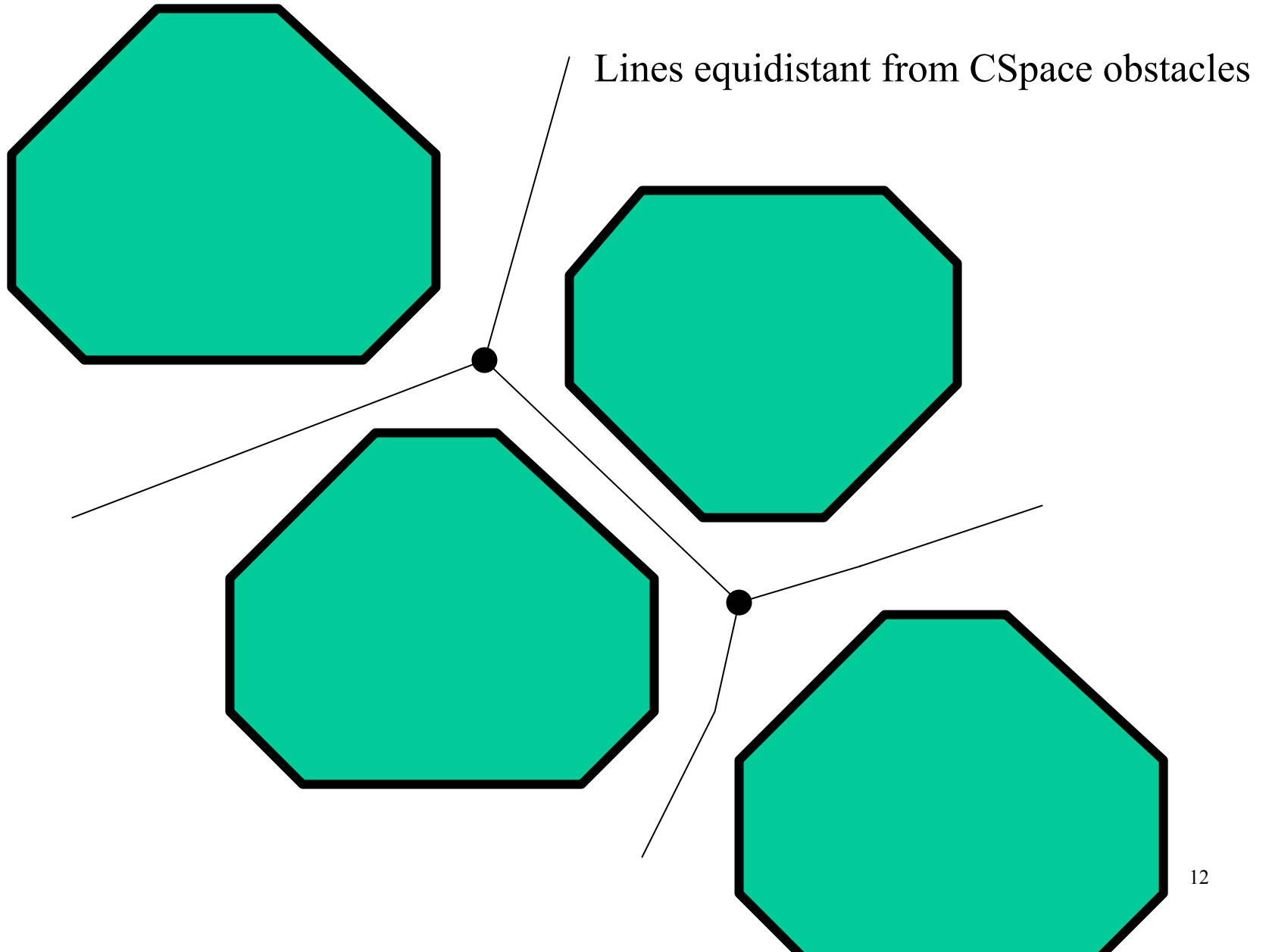
What are some other types of roadmaps?

Start
position

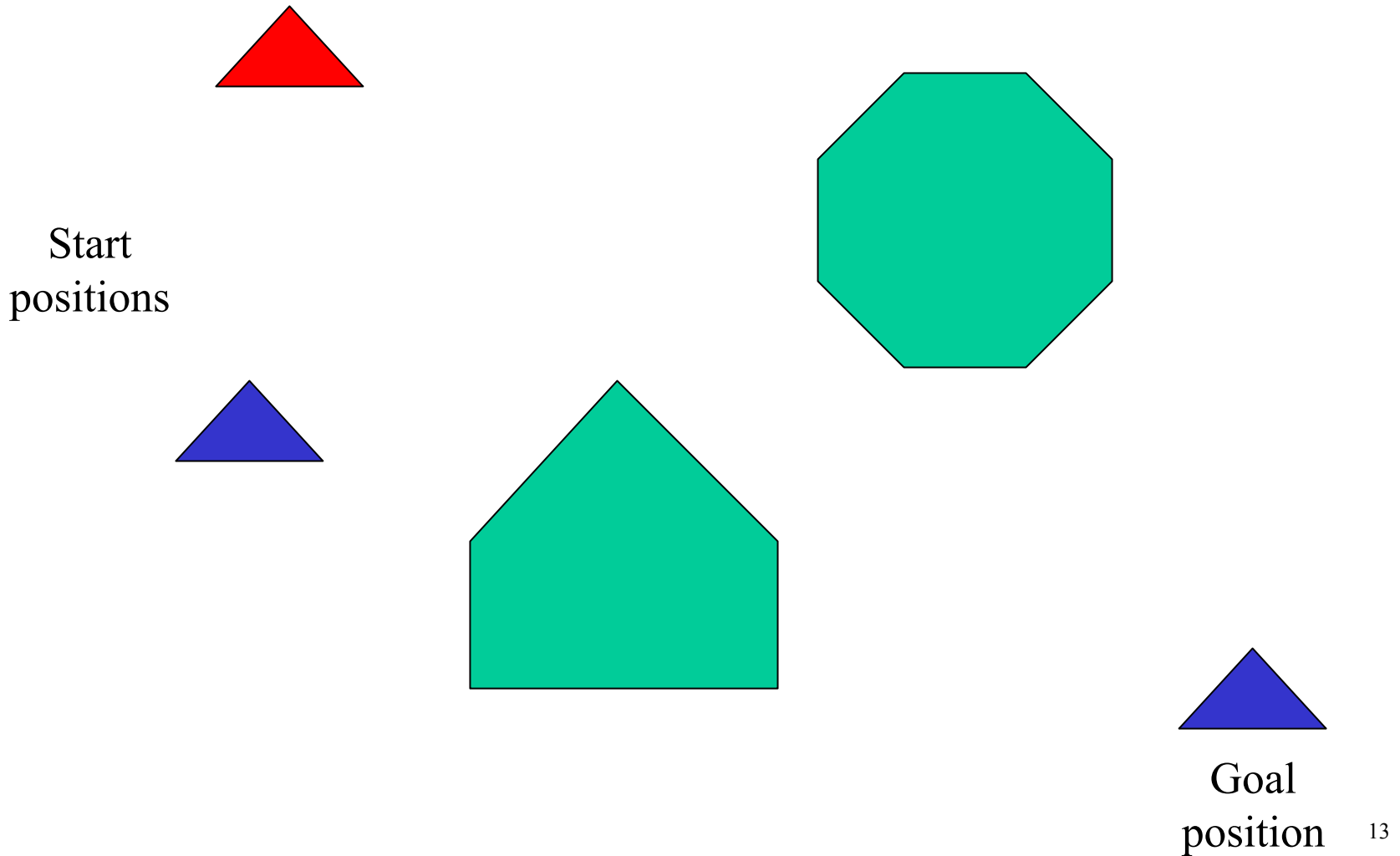


Goal
position

Voronoi Diagrams



Path Planning With an *Adversary*



Types of Competitive Games

Two Player and Multi-Player

Zero and non-zero sum games

$$f(\text{player1}) = - f(\text{player2})$$

Perfect and imperfect information

Stochastic games

➔ Two player, zero sum games with perfect info

Two Player Games With Perfect Information

- Initial State
- Successor Function
- Terminal Test
- Utility Function

Tic Tac Toe

X		
	X	
		O

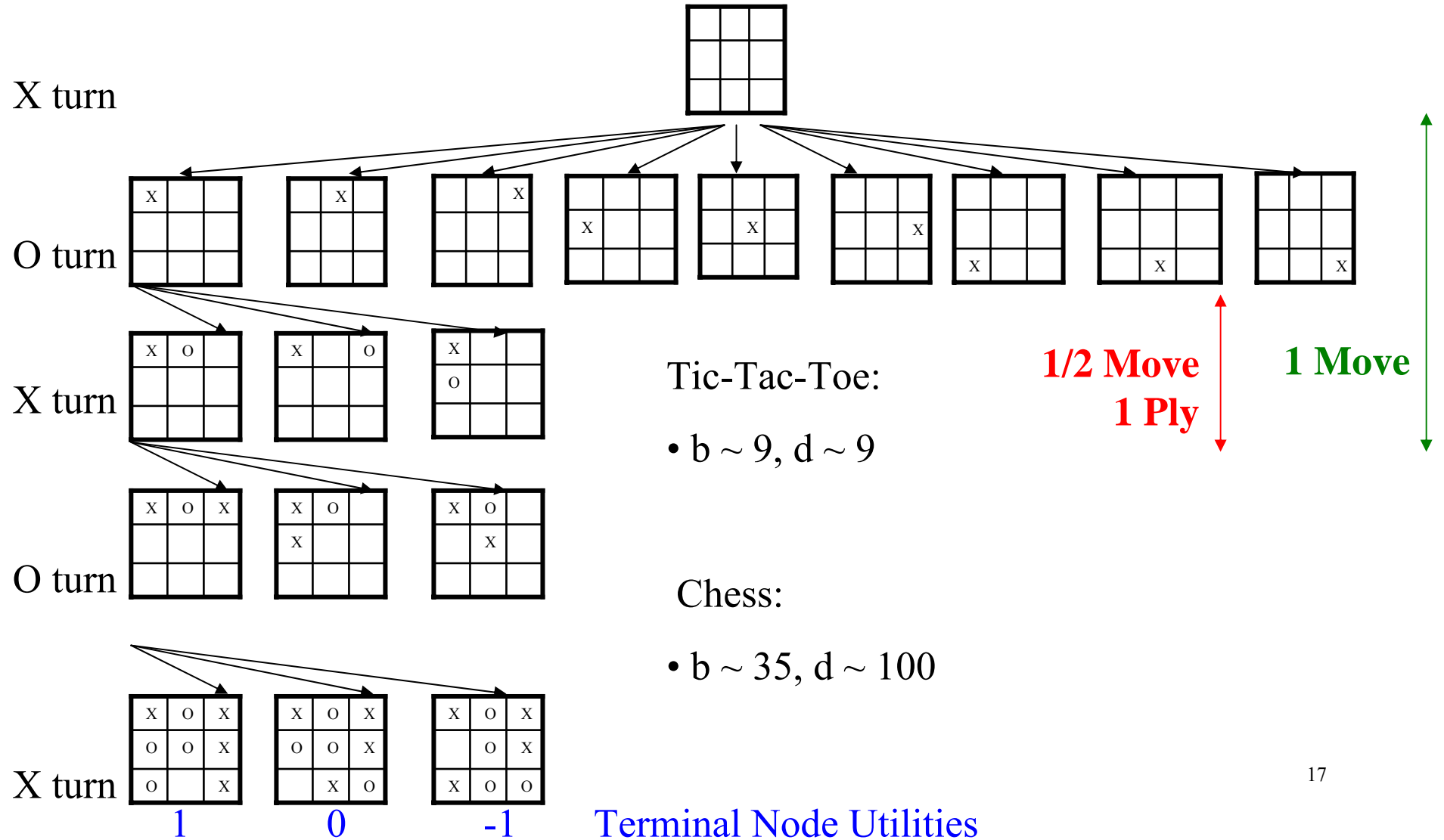
Two Player Games With Perfect Information

- Initial State
 - Empty board
- Successor Function
 - Place X or O in empty square
- Terminal Test
 - Three X's or O's in a line is a win
 - Else no empty squares is a tie
- Utility Function
 - 1 for win
 - 0 for tie
 - -1 for lose

Tic Tac Toe

X		
	X	
		O

Game Tree



Optimal Strategies

- Strategy π : state \rightarrow move
- Optimal strategy π^* :
 - One that performs at least as well as any other strategy.

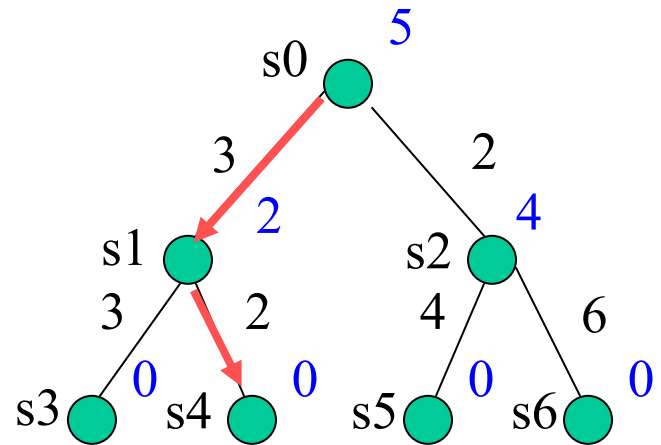
Optimal Strategies: What is an optimal strategy for informed search?

An optimal strategy $\pi^*(n)$ selects the branch to the sub tree containing the path with minimum cost V^* .

$$V^*(n) = \text{Min}_{c \text{ in children}(n)} [P(c,n) + V^*(c)]$$

$V^*(n) = 0$ if n is a terminal node

$$\pi^*(n) = \text{arg Min}_{c \text{ in children}(n)} [P(c,n) + V^*(c)]$$



For games we will use $V^*(n)$ to denote utility (reward) rather than cost.

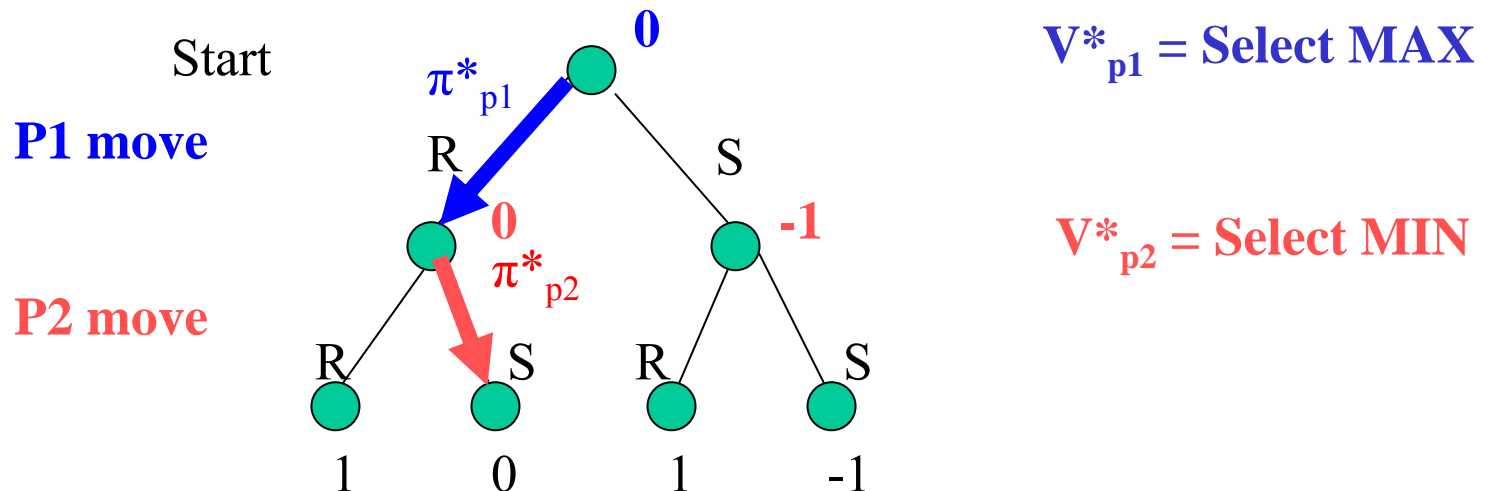
Optimal Strategies for 2 Person Games

Assume opponent j uses an optimal strategy π^*_j .

Player i strategy π^*_{pi} selects branch to sub-tree containing the best state $f_{pi}(s)$ reachable using π^*_j .

Assume zero sum game $f_{p1} = -f_{p2}$.

Player 1 maximizes, while player 2 minimizes.



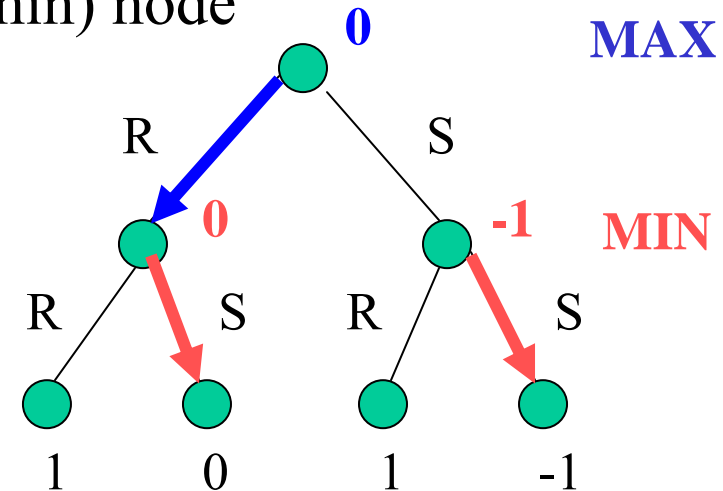
Optimal Strategy: Min-Max

Utilities:

- $V^*_{p_i}(n) = \text{Utility}(n)$ if n is a terminal node
- $V^*_{p_1}(n) = \text{Max}_{s \text{ in successors}(n)} V^*_{p_2}(s)$ if n is a p_1 (max) node
- $V^*_{p_2}(n) = \text{Min}_{s \text{ in successors}(n)} V^*_{p_1}(s)$ if n is a p_2 (min) node

Strategies:

- $\pi^*_{p_1}(n) = \arg \text{Max}_{s \text{ in successors}(n)} V^*_{p_2}(s)$
- $\pi^*_{p_2}(n) = \arg \text{Min}_{s \text{ in successors}(n)} V^*_{p_1}(s)$



Compute using depth first search

$O(b^{m+1})$ time

$O(b^*m)$ space

MiniMax-Decision(state) **returns** *an action*

inputs: *state*, current state in game

$v \leftarrow \text{Max-Value}(\text{state})$

return the *action* in $\text{Successors}(\text{state})$ with value v

Function $\text{Max-Value}(\text{state})$ **returns** a *utility value*

if $\text{Terminal-Test}(\text{state})$ **then return** $\text{Utility}(\text{state})$

$v \leftarrow -\infty$

for a, s in $\text{Successors}(\text{state})$ **do**

$v \leftarrow \text{Max}(v, \text{Min-Value}(s))$

return v

Function $\text{Min-Value}(\text{state})$ **returns** a *utility value*

if $\text{Terminal-Test}(\text{state})$ **then return** $\text{Utility}(\text{state})$

$v \leftarrow -\infty$

for a, s in $\text{Successors}(\text{state})$ **do**

$v \leftarrow \text{Min}(v, \text{Max-Value}(s))$

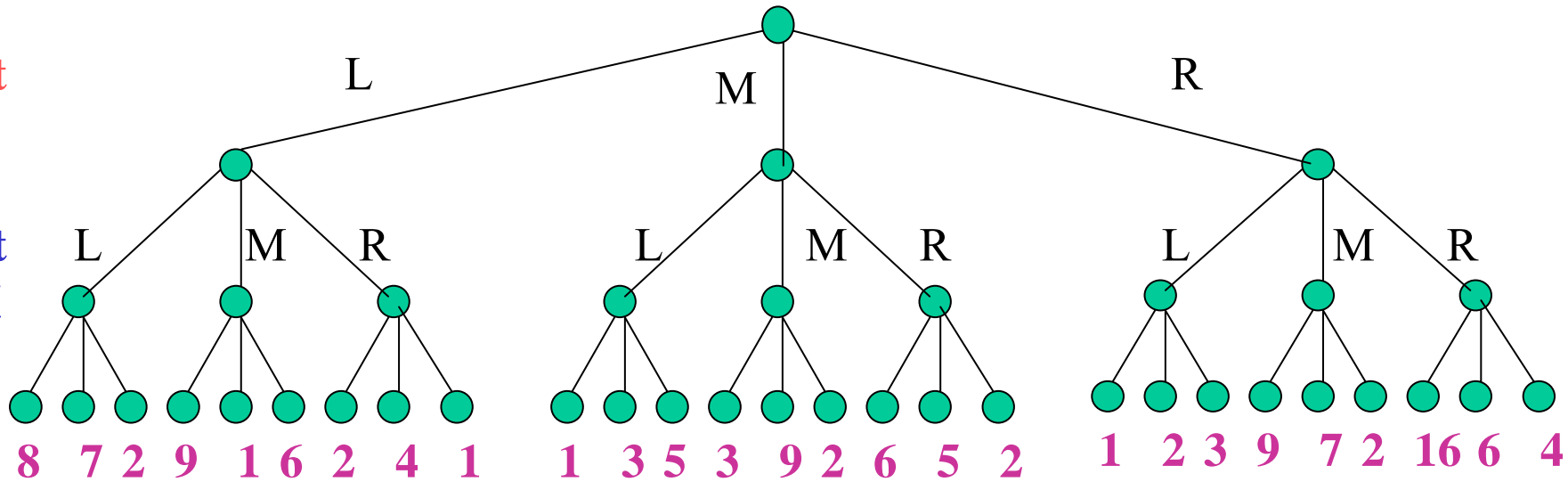
return v

Compute Mini-Max

Select
MAX

Select
MIN

Select
MAX

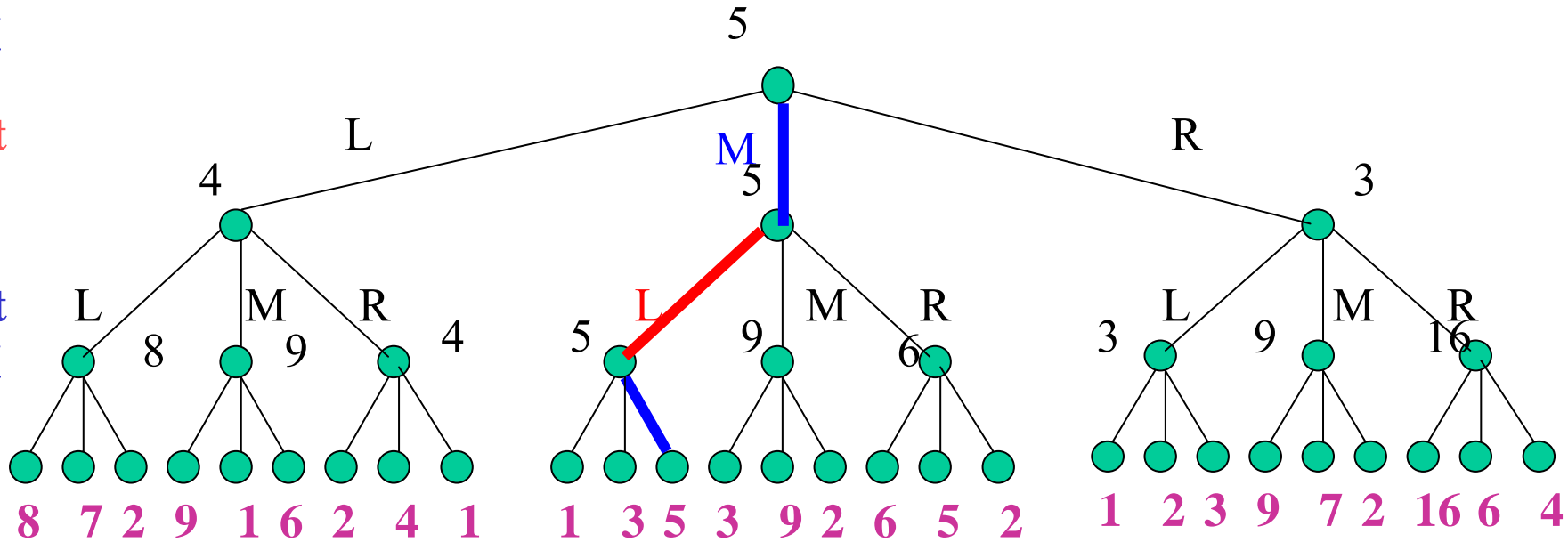


Compute Mini-Max

Select
MAX

Select
MIN

Select
MAX



Chess using 2Ghz PC

- Min/Max 5 ply, 2 ½ moves
- Expert 10 ply, 5 moves
- Average Game 100 ply 50 moves

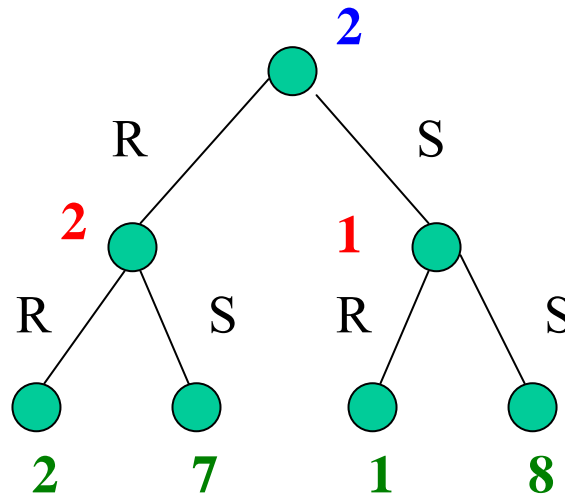
→ Terminal nodes are too far to reach.

Heuristic Evaluation Fn

- Trees are too large to explicitly enumerate all branches (Chess $\sim 35^{101}$).

\Rightarrow Evaluation estimates board quality

(pawn = 1, knight/bishop = 3, rook = 5, queen 9).



Select MAX

Select MIN

Estimates from
evaluation

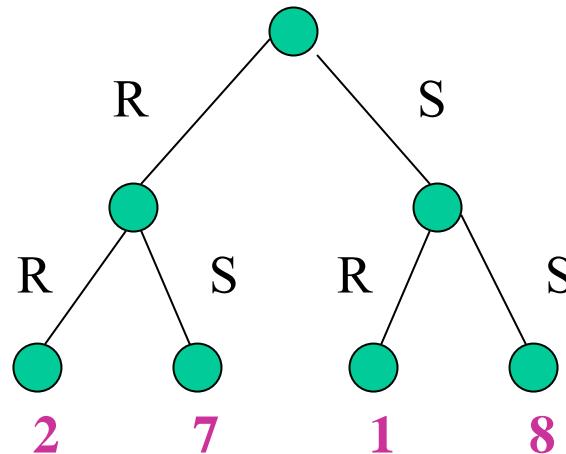
Alpha-Beta Pruning

- Use depth-first to search for best option.
- Trim an option as soon as we show another is better.

MAX

MIN

MAX



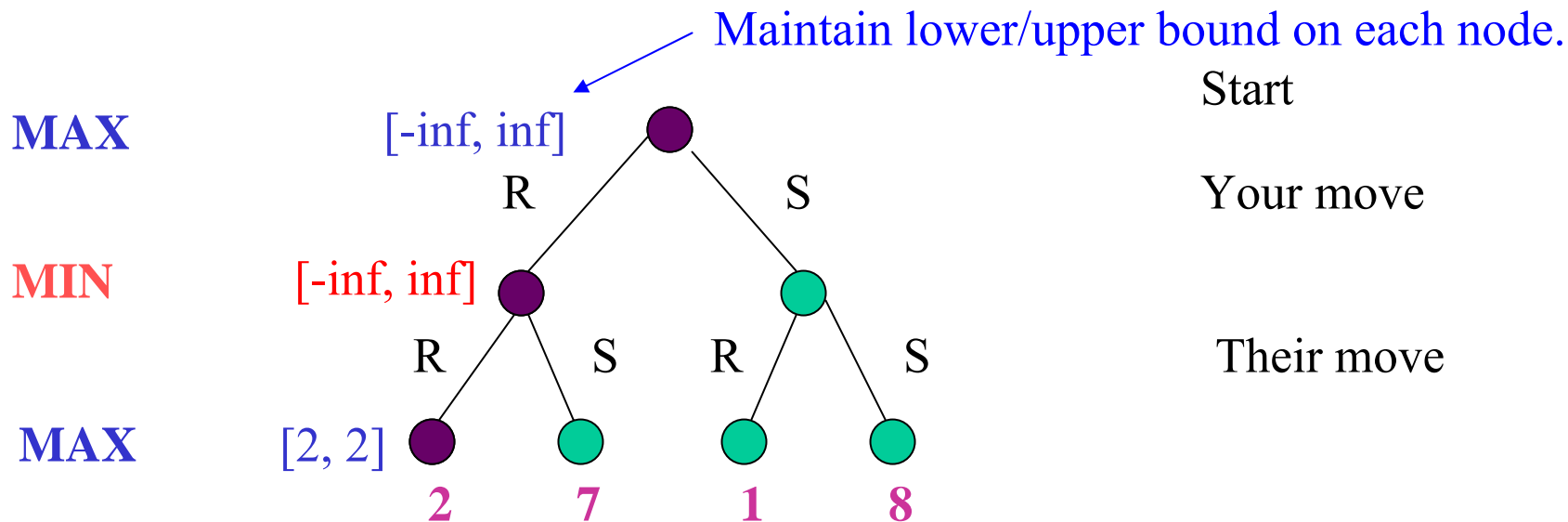
Start

Your move

Their move

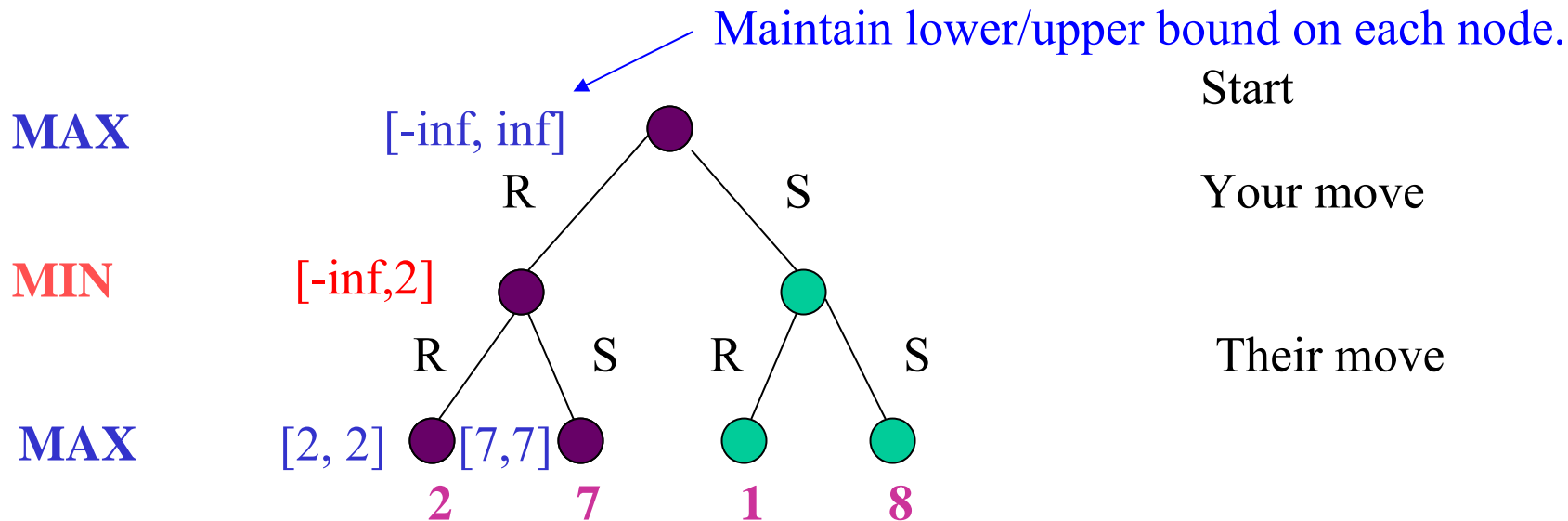
Alpha-Beta Pruning

- Use depth-first to search for best option.
- Trim an option as soon as we show another is better.



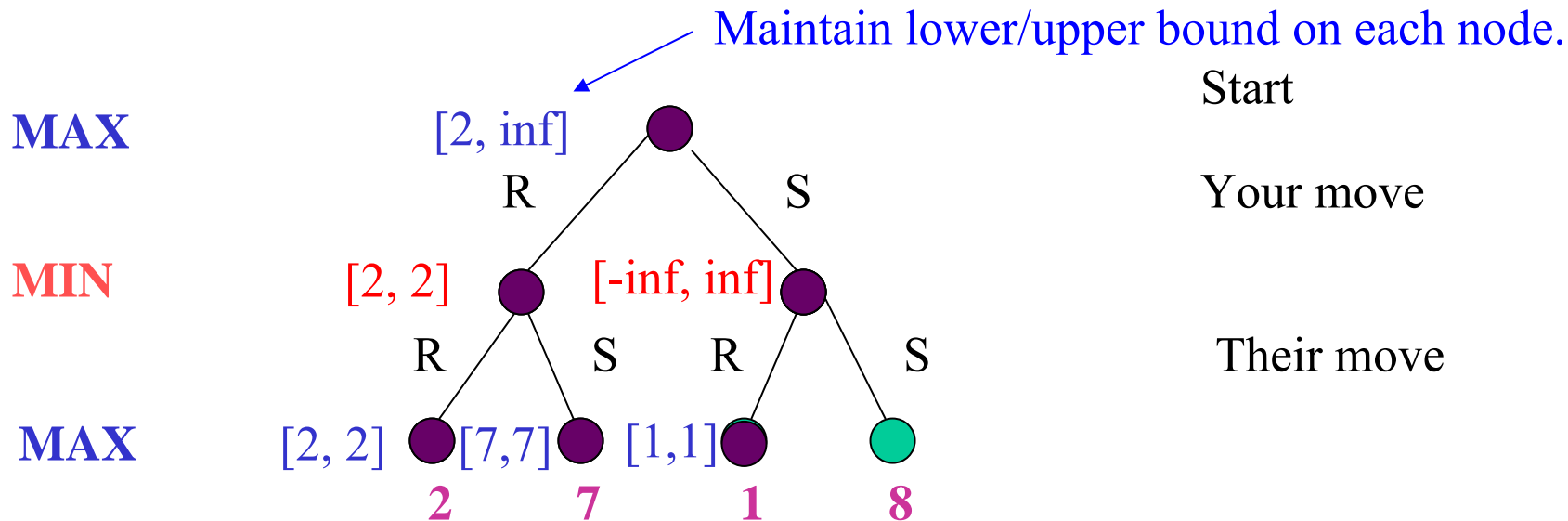
Alpha-Beta Pruning

- Use depth-first to search for best option.
- Trim an option as soon as we show another is better.



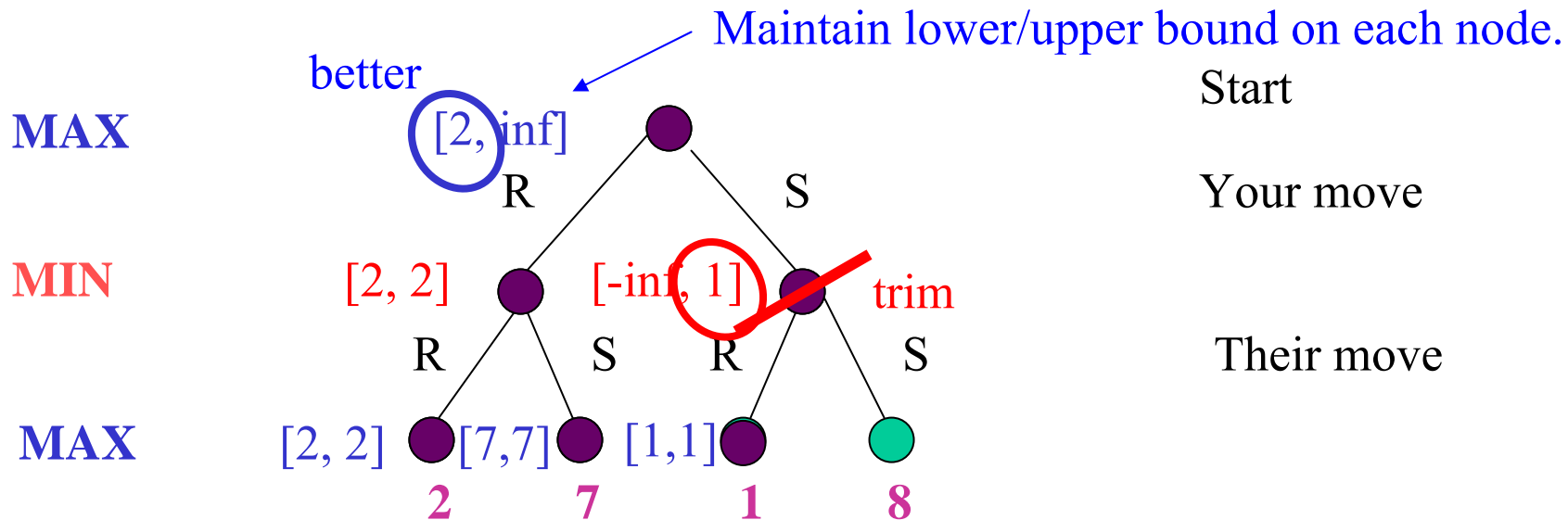
Alpha-Beta Pruning

- Use depth-first to search for best option.
- Trim an option as soon as we show another is better.



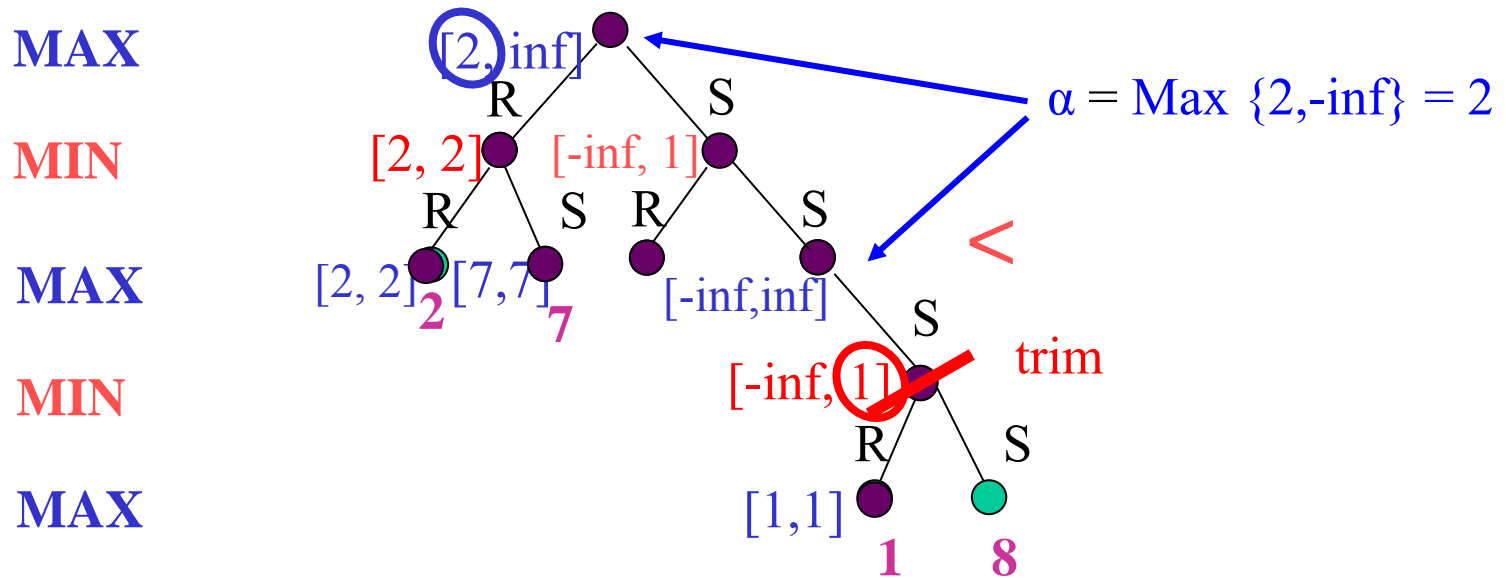
Alpha-Beta Pruning

- Use depth-first to search for best option.
- **Trim an option** as soon as we show **another is better**.



Alpha of Alpha-Beta Pruning

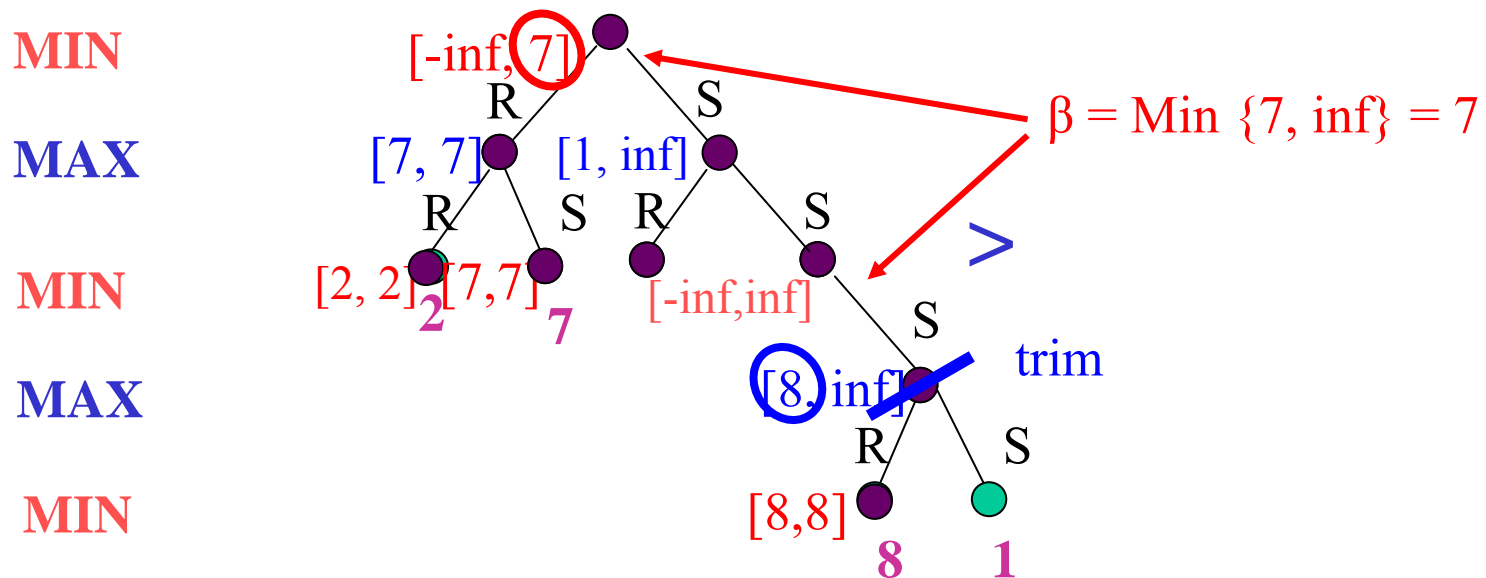
- The better option may be anywhere in the tree.



- Let α be the highest value choice found so far for any choice point of Max (a greatest lower bound).
 - $\alpha = \text{Max}$ [lower bounds of previous max nodes]
- If α is above the upper bound for a node n of Min, prune n .

Beta of Alpha-Beta Pruning

- The argument applies equally when the roles are reversed.



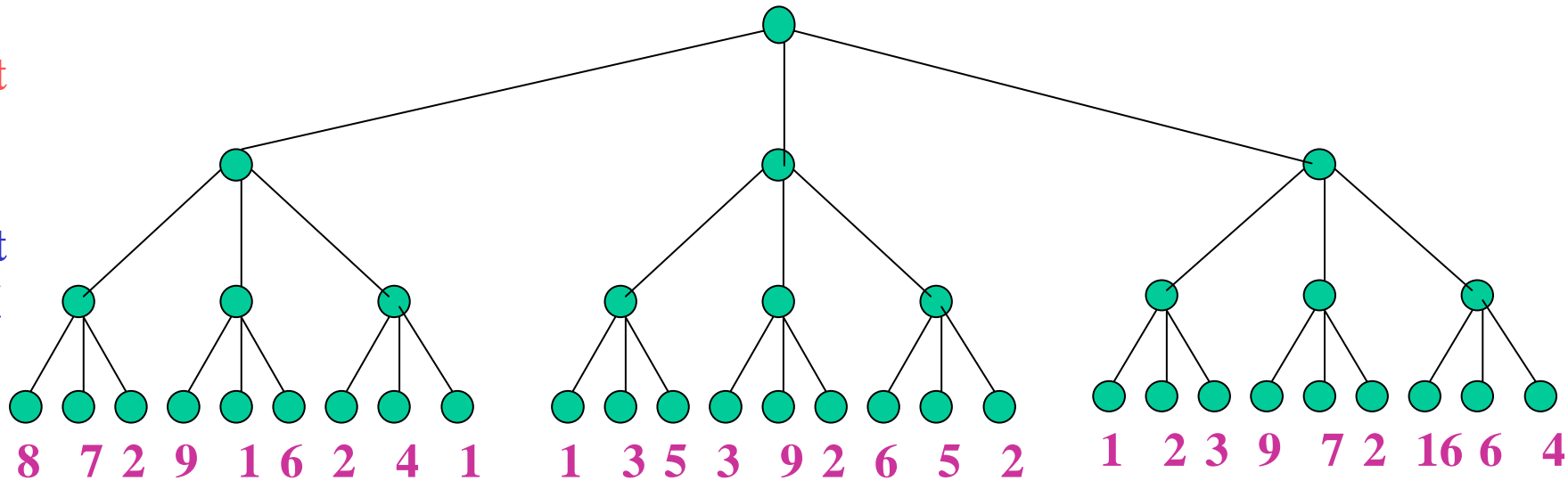
- Let β be the **lowest value choice** found so far for **any choice point of Min** (a least upper bound).
 - $\beta = \text{Min}$ [upper bounds of previous min nodes]
- If β is **below** the **lower bound** for a node **n** of **Max**, prune **n**.

Putting it Together: Alpha-Beta Pruning

Select
MAX

Select
MIN

Select
MAX



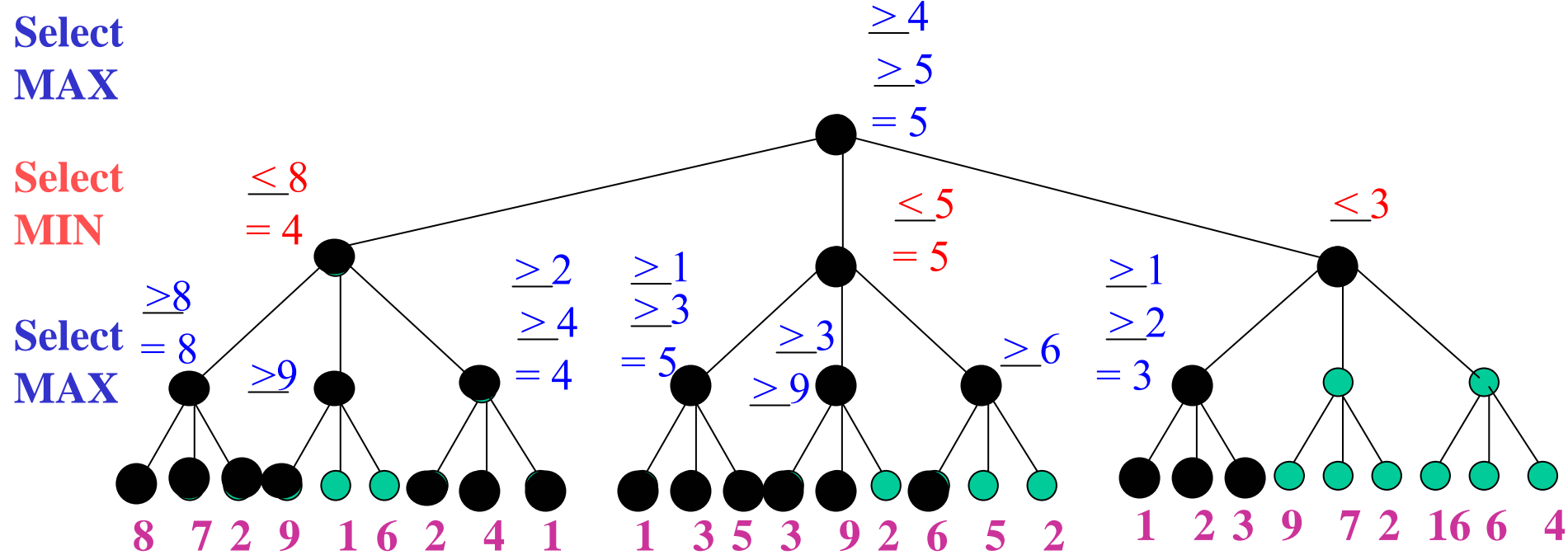
Cut a Min node if $< \alpha$:

Cut a max node if $> \beta$:

α = Max [lower bounds of previous max nodes]

β = Min [upper bounds of previous min nodes]

Putting it Together: Alpha-Beta Pruning



Cut a Min node if $\alpha <$: 8
9

Cut a max node if $> \beta$: 8
4

α = Max [lower bounds of previous max nodes]

β = Min [upper bounds of previous min nodes]

Function Alpha-Beta-Search(*state*) **returns** an action
inputs: *state*, current state in game
 $v \leftarrow \text{Max-Value}(\text{state}, -\infty, +\infty)$
return the *action* in Successors(*state*) with value v

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**
 $v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta))$
 if $v \leq \beta$ **then return** v
 $\alpha \leftarrow \text{Max}(\alpha, v)$
return v

Function Min-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**
 $v \leftarrow \text{Min}(v, \text{Max-Value}(s, \alpha, \beta))$
 if $v \leq \alpha$ **then return** v
 $\beta \leftarrow \text{Min}(\beta, v)$
return v

From AIMA Figure 6.7

Chess using 2Ghz PC

- Min/Max 5 ply, 2 ½ moves
- Expert 10 ply, 5 moves
- Alpha/Beta 10 ply, 5 moves
- Grand Master
- Deep Blue 30 RS/6000 + 480 VLSI Chips
 - 40 ply using supercomputer
 - 4000 openings, 6 piece endgames
 - 8,000 feature evaluation,
 - 700,000 grandmaster games for consensus

Best n forward pruning

- At each level consider only the ‘n’ ‘most promising’ choices.
 - At each level apply the heuristic evaluation function.
 - Sort the positions according to the heuristic evaluation.
 - Discard all but the best ‘n’.
- Puts an upper bound on the branching factor.
 - Advantage: Can look ahead more ply this way.
 - Disadvantage: Doesn’t see good moves that initially look bad such as sacrifices.