Search: overview

Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

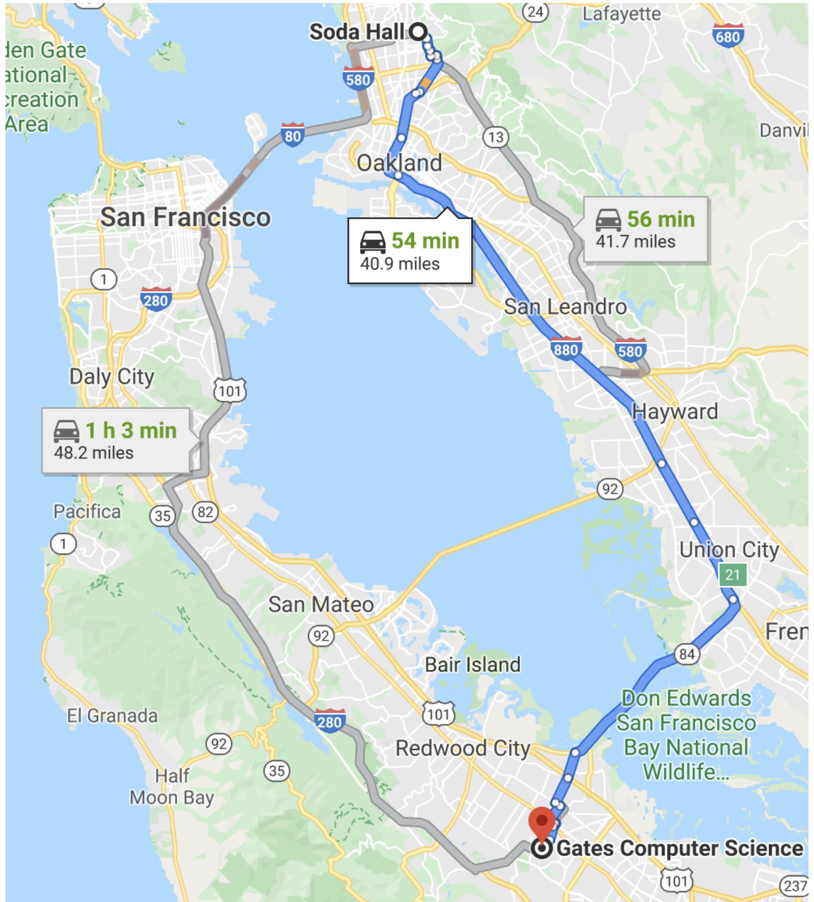
Bayesian networks

Reflex **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

Machine learning

Application: route finding



Objective: shortest? fastest? most scenic?

Actions: go straight, turn left, turn right

Application: robot motion planning

Objective: fastest path

Actions: acceleration and throttle

Application: robot motion planning

Objective: fastest? most energy efficient? safest? most expressive?

Actions: translate and rotate joints

Application: multi-robot systems

Objective: fastest? most energy efficient?

Actions: acceleration and steering of all robots

Application: machine translation

la maison bleue

$\downarrow$

the blue house

Objective: fluent English and preserves meaning

Actions: append single words (e.g., `the`)

Beyond reflex

Classifier (reflex-based models):

$x \longrightarrow \boxed{f} \longrightarrow$ single action $y \in \{-1, +1\}$

Search problem (state-based models):

$x \longrightarrow \boxed{f} \longrightarrow$ **action sequence** $(a_1, a_2, a_3, a_4, \dots)$

Key: need to consider future consequences of an action!

Roadmap

Modeling

Modeling Search Problems

Algorithms

Tree Search

Dynamic Programming

Uniform Cost Search

Programming and Correctness of UCS

A*

A* Relaxations

Learning

Structured Perceptron

Paradigm

Modeling

Inference

Learning

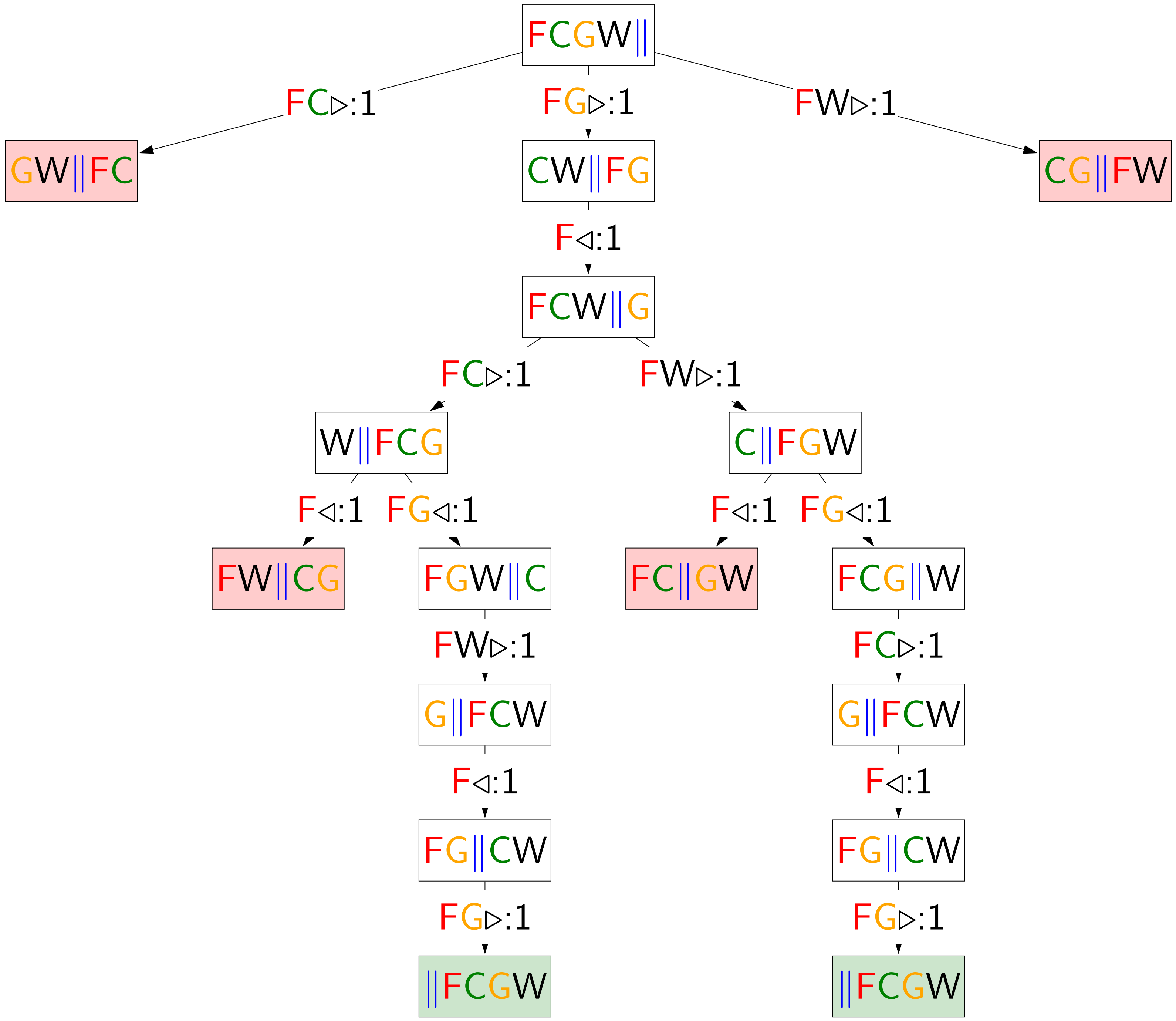
Search: modeling

Farmer Cabbage Goat Wolf

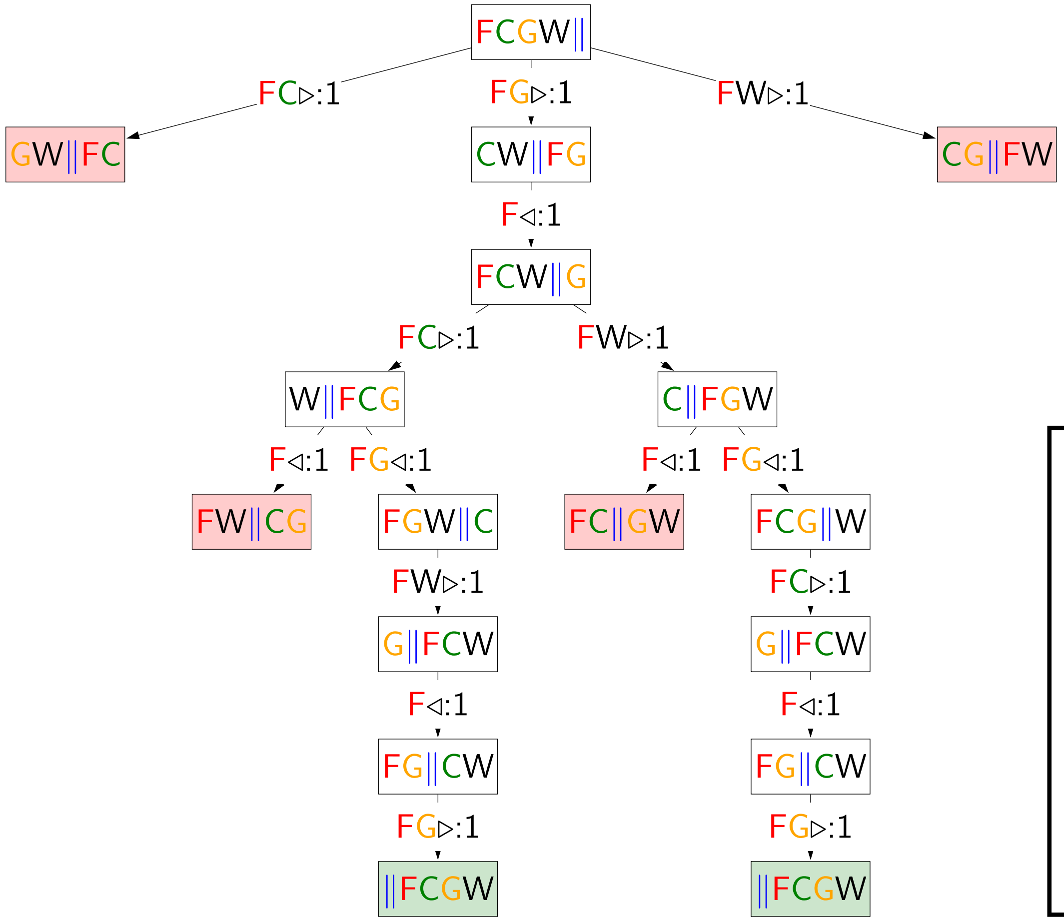
Actions:

F$\triangleright$	F$\triangleleft$
FC$\triangleright$	FC$\triangleleft$
FG$\triangleright$	FG$\triangleleft$
FW$\triangleright$	FW$\triangleleft$

Approach: build a **search tree** ("what if?")


FCGW||
FC▷:1
FG▷:1
FW▷:1
GW||FC
CW||FG
CG||FW
F◁:1
FCW||G
FC▷:1
FW▷:1
W||FCG
C||FGW
F◁:1
FG◁:1
F◁:1
FG◁:1
FW||CG
FGW||C
FC||GW
FCG||W
FW▷:1
FC▷:1
G||FCW
G||FCW
F◁:1
F◁:1
FG||CW
FG||CW
FG▷:1
FG▷:1
||FCGW
||FCGW

Search problem



Definition: search problem

- s_{start}: starting state
- Actions(s): possible actions
- Cost(s, a): action cost
- Succ(s, a): successor
- IsEnd(s): reached end state?

Transportation example

Example: transportation

Street with blocks numbered 1 to n.

Walking from s to $s+1$ takes 1 minute.

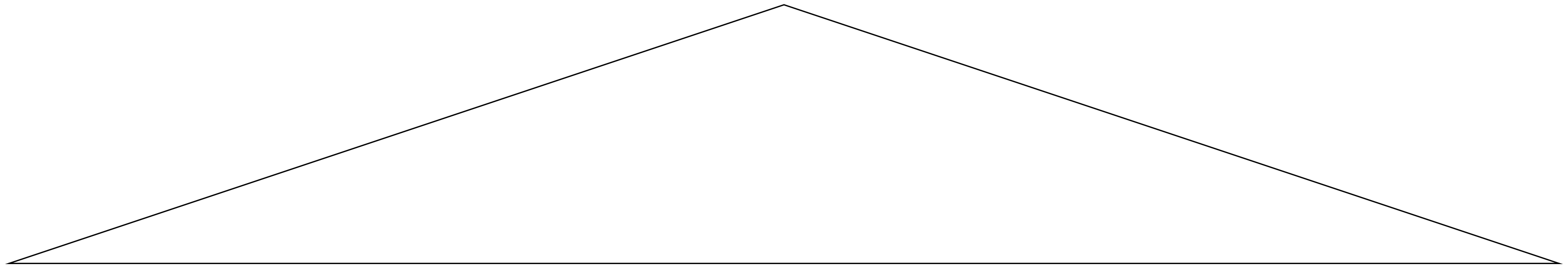
Taking a magic tram from s to $2s$ takes 2 minutes.

How to travel from 1 to n in the least time?

[semi-live solution: `TransportationProblem`]

Search: tree search

Backtracking search



[whiteboard: search tree]

If b actions per state, maximum depth is D actions:

- Memory: $O(D)$ (small)
- Time: $O(b^D)$ (huge) $[2^{50} = 1125899906842624]$

Backtracking search

Algorithm: backtracking search

def backtrackingSearch(s, path):
    If IsEnd(s): update minimum cost path
    For each action $a \in$ Actions(s):
        Extend path with Succ(s, a) and Cost(s, a)
        Call backtrackingSearch(Succ(s, a), path)
    Return minimum cost path

[semi-live solution: `backtrackingSearch`]

Depth-first search

Assumption: zero action costs

Assume action costs $\text{Cost}(s, a) = 0$.

Idea: Backtracking search + stop when find the first end state.

If b actions per state, maximum depth is D actions:

- Space: still $O(D)$
- Time: still $O(b^D)$ worst case, but could be much better if solutions are easy to find

Breadth-first search

Assumption: constant action costs

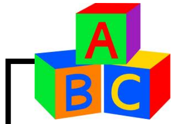
Assume action costs $\text{Cost}(s, a) = c$ for some $c \geq 0$.

Idea: explore all nodes in order of increasing depth.

Legend: b actions per state, solution has d actions

- Space: now $O(b^d)$ (a lot worse!)
- Time: $O(b^d)$ (better, depends on d, not D)

DFS with iterative deepening



Assumption: constant action costs

Assume action costs $\text{Cost}(s, a) = c$ for some $c \geq 0$.

Idea:

- Modify DFS to stop at a maximum depth.
- Call DFS for maximum depths $1, 2, \ldots$.

DFS on d asks: is there a solution with d actions?

Legend: b actions per state, solution size d

- Space: $O(d)$ (saved!)
- Time: $O(b^d)$ (same as BFS)

Tree search algorithms

Legend: b actions/state, solution depth d, maximum depth D

Algorithm	**Action costs**	**Space**	**Time**
Backtracking	any	$O(D)$	$O(b^D)$
DFS	zero	$O(D)$	$O(b^D)$
BFS	constant ≥ 0	$O(b^d)$	$O(b^d)$
DFS-ID	constant ≥ 0	$O(d)$	$O(b^d)$

- Always exponential time
- Avoid exponential space with DFS-ID

Tree Search Review