Help

Search: deterministic state space models

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

Review: methodology

Task: specified by environment e and utility function U

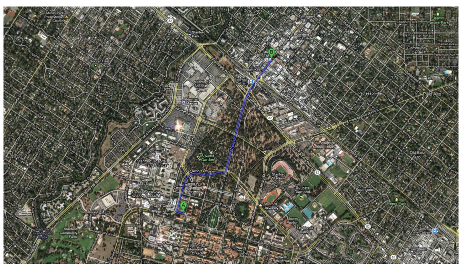
Rational agent: $A_{\text{opt}} = \arg\max_{A \in \text{Agents}} \mathbb{E}[U(A, e)]$

Issue: can't achieve because lack of computation or information

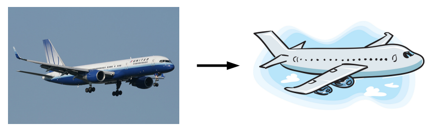
Modeling: build simplified environment e' and utility function U'

Rational agent: $A_{\text{opt}}{}' = \arg\max_{A' \in \text{Agents}'} \mathbb{E}[U'(A', e')]$

Real-world problem: route finding



Preferences: shortest? fastest? most scenic?
Constraints: traffic lights? pedestrians? construction?
Solution: a plan sequence of actions that achieves the goal

Modeling 

Outline

- **Definition of deterministic state space models**
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

Key concept: state

Definition: State

A **state** contains all information about agent/environment that are (i) non-constant and are (ii) relevant to the task.

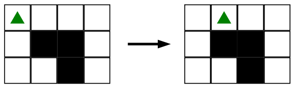
Example:

Position: 35.67,120.63; Orientation: 50°; Velocity: 30mph
Position of other objects: ...
Date/time: Sat Oct 06 2012 09:42:42 GMT-0700 (PDT)
Value of π: 3.14159265...
Price of gold: $1764.90/oz
...

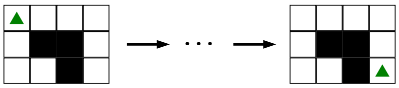
Modeling: involves deciding what to include in a state

A simple model of route finding

Actions: move to adjacent squares (discrete steps)



Goal: end up in bottom-right square



Assumption: environment is deterministic
Know exactly how actions affect the environment.

General formulation

Definition: Deterministic state space model

State: $s \in \text{States}$

Action: $a \in \text{Actions}(s)$

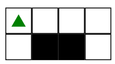
Successor: $\text{Succ}(s, a) \in \text{States}$

Cost: $\text{Cost}(s, a) \in \mathbb{R}$

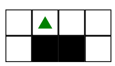
Start state: $s_{\text{start}} \in \text{States}$

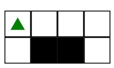
Goal test: $\text{IsGoal}(s)$

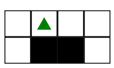
Simple model for route finding

$\text{States} \ni s_{\text{start}} =$ 

$\text{Actions}(s_{\text{start}}) \ni a = \text{East}$

$\text{Succ}(s_{\text{start}}, a) =$ 

$\text{Cost}($  $, \text{East}) = 1$

$\text{IsGoal}($  $) = \text{false}$

General formulation

Definition: Optimization problem

Find a **path** (sequence of actions) $p = (a_1, \dots, a_n)$ with minimum path cost:

$$\text{PathCost}(p) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \text{Cost}(s_{i-1}, a_i) \text{ if } p \text{ reaches the goal:}$$

$$[s_0 = s_{\text{start}}, s_i = \text{Succ}(s_{i-1}, a_i), \text{IsGoal}(s_n) = \text{true}]$$

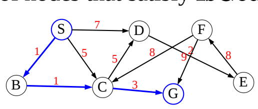
$$= \infty \text{ otherwise.}$$

State space graphs

Each **node** is a state $s \in \text{States}$

Each **(directed) edge** is a pair $(s, \text{Succ}(s, a))$ with cost $\text{Cost}(s, a)$ for action $a \in \text{Actions}(s)$

Goal nodes: subset of nodes that satisfy IsGoal



Optimization problem: find a path from start node to a goal node with minimum cost

Don't need to construct full graph explicitly in code.

Agents, environments, utilities

Environment: $e = (\text{States}, \text{Actions}, \text{Succ}, \text{Cost}, s_{\text{start}}, \text{IsGoal})$

Agent: A takes an environment e and returns a path p

Utility: $U(A, e) = -\text{PathCost}(A(e))$

Rational agent: solves optimization problem on $(e, \text{PathCost})$

Outline

- Definition of deterministic state space models
- **Motivating applications**
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

Application: robot navigation

Task: have robot transport object from one place to another

Model:

- State: position, orientation, joint angles, whether grasping object
- Actions: flex/rotate joints, activate wheels
- Cost: energy/time consumed, penalty if bump into something
- Goal test: whether object is in desired place

Application: machine translation

Task: translate English to French

the blue house

↓

la maison bleue

Simple model:

- State: English words translated so far E
- Actions: choose English word $e \notin E$, French word f
- Cost: $-\textbf{Fidelity}(e, f)$
- Goal test: Whether E covers whole English sentence

Application: machine translation

Task: translate French to English

the blue house

↓

la maison bleue

Improved model:

- State: English words translated so far E + last French word f'
- Actions: choose English word $e \notin E$, French word f
- Cost: $-\textbf{Fidelity}(e, f) - \textbf{Fluency}(f', f)$
- Goal test: Whether E covers whole English sentence

Application: software/hardware verification

Task: ensure systems can't do bad stuff (e.g., dereference null pointers, buffer overflow, leak sensitive information)

Model:

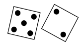
- State: program state (program counter, register contents, etc.)
- Actions: external inputs from user
- Goal test: whether program state violates a specification

Note: want **absence** of paths - everything is turned upside down!

Outline

- Definition of deterministic state space models
- Motivating applications
- **Some challenges: uncertainty and continuous spaces**
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

Unknown costs



In practice, don't know exact edge costs (e.g., traffic).

Assumption: Random edge costs

Each edge cost is a random variable:

$$\text{Cost}(s,a) = \begin{cases} \text{LowCost}(s,a) & \text{with probability } \alpha \\ \text{UpCost}(s,a) & \text{with probability } 1-\alpha \end{cases}$$

Rational agent:

$U(A,e) = -\text{PathCost}(A(e)) = -\sum_{i=1}^n \text{Cost}(s_{i-1},a_i)$

$\mathbb{E}[U(A,e)] = -\sum_{i=1}^n \mathbb{E}[\text{Cost}(s_{i-1},a_i)]$

$\mathbb{E}[\text{Cost}(s,a)] = \alpha\,\text{LowCost}(s,a) + (1-\alpha)\text{UpCost}(s,a)$

Rational agent same as with deterministic costs $\mathbb{E}[\text{Cost}(s,a)]$!

Unknown costs

States: $s_0, \ldots, s_{1000}$

Three paths from s_0 to s_{1000}:

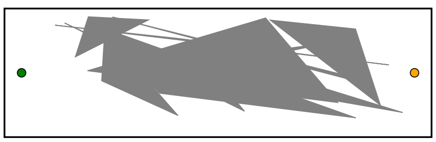
- Cost of 1000 with certainty
- Cost of 0 or 2000, each with probability $\frac{1}{2}$
- Cost of 0 or 2, repeated 1000 times independently

Same expected utility? Only if utility is linear in edge costs.

What if utility were 1 if path cost at most 900 and 0 otherwise?

What if traversing edge fails with probability $\frac{1}{2}$? Need MDPs!

Continuous state spaces



- States: all points $(x,y) \in [0,100]^2$
- Actions: move in any direction by any distance

Infinite!

Discretization:

- States: corner points of the polygons
- Actions: move in straight line to another corner point that doesn't intersect rubble

Outline

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- **Algorithms**
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

Review: methodology

Model: environment e and utility function U

Rational agent specification: $A_{\text{opt}} \in \arg\max_A \mathbb{E}[U(A,e)]$

Deterministic state space **model** ("graph with edge costs"):

- Environment $e = (\text{States}, \text{Actions}, \text{Succ}, \text{Cost}, s_{\text{start}}, \text{IsGoal})$
- Utility $U(A,e) = -\text{PathCost}(A(e))$

Rational agent specification: $A_{\text{opt}}(e) \in \arg\min_p \text{PathCost}(p)$

Agent implementations (**algorithms**): DAG search, DFS, BFS, UCS, A*, Bellman-Ford (rational? depends on model)

Review: modeling

Example task:

- Traversing one east-west block is 3 time units; north-south is 1
- Interections: left (2 time units), straight (1), right (0)
- Want to get from point a to b in Manhattan
- Make no more than k left turns
- Minimize commute time

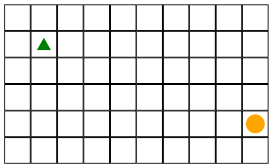
5 easy steps:

- Write down possible agent outputs.
- Break down output into sequence of actions.
- Write down path cost (including constraints).
- Add things to state to enable calculation of path cost.
- Choose algorithm based on model.

Different algorithms for different models

Algorithm	**Allow cycles?**	**Edge costs**	**Use case**
DAG search	no	anything	MT, speech
DFS	yes	$= 0$	verification
BFS	yes	$= \text{constant}$	simple route finding
UCS, A*	yes	≥ 0	route finding
Bellman-Ford	yes	anything	handle rewards

Analytic solutions



Exploit special structure to find optimal path analytically if possible.

Deterministic state space models don't always require search.

Outline

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - **DAG search, DFS**
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

Directed acyclic graphs

Assumption: Acyclicity
State space graph has no (directed) cycles.

Intuition: every action makes progress towards the goal state.

Example: machine translation

Example:

Notes

Directed acyclic graphs

Compute $\text{BackCost}(s)$, the minimum cost from s to any goal state, **recursively**:

$$\text{BackCost}(s) = \begin{cases} 0 & \text{if } s = s_{\text{goal}} \\ \min_{a \in \text{Actions}(s)} [\text{Cost}(s,a) + \text{BackCost}(\text{Succ}(s,a))] & \text{otherwise.} \end{cases}$$

Algorithm: DAG search

```
backCost = {} # state s -> minimum cost from s to goal
def GetBackCost(s):
  if s == goalState: return 0
  if backCost[s] != None: return backCost[s] # Use memoization
  backCost[s] = float('inf')
  for a in Actions(s):
    t = Succ(s, a) # Try going from s to t
    backCost[s] = min(backCost[s], Cost(s, a) + GetBackCost(t))
  return backCost[s]
GetBackCost(startState)
```


Cyclic graphs, zero costs

What if there are cycles?

Assumption: Zero costs
All edge costs are zero ($\text{Cost}(s,a) = 0$ for all s, a).

Strategy: Traverse edges in an arbitrary order until we find a goal state.

Cyclic graphs, zero costs

Algorithm: Depth-first search (DFS)

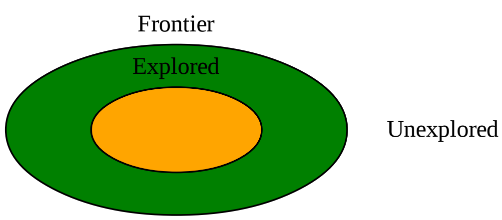
```
explored = set()
def DFS(path, s):
  if IsGoal(s): return path
  for a in Actions(s):
    t = Succ(s, a) # Try going from s to t
    if t in explored: continue # Avoid cycles
    explored.add(t)
    path = DFS(path + [a], t)
    if path != None: return path # Found
  return None # Not found
DFS([], set([startState]), startState)
```

Complexity: $O(\text{number of edges})$

Outline

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - **Uniform cost search, BFS**
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

High-level strategy



Keep track of:

- Explored: nodes we're done with
- Frontier: nodes we're seen, figuring out how to get there cheaply
- Unexplored: nodes we haven't seen

Forward and backward costs

Definition: Backward costs
Let $\text{BackCost}(s)$ be the minimum cost from s to any goal state.

DAG search computes all $\text{BackCost}(s)$ recursively.

Definition: Forward costs
Let $\text{ForwCost}(s)$ be the minimum cost from s_{start} to s.

Uniform cost search (Dijkstra's algorithm) computes $\text{ForwCost}(s)$ in increasing order.

Uniform cost search (UCS)

Algorithm: Uniform cost search

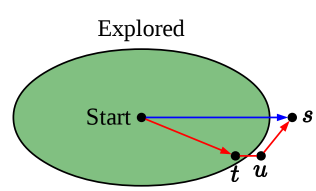
```
explored = set()
frontier = PriorityQueue()
frontier.update(initState, 0)
while True:
  if frontier.size() == 0: return None
  s, priority = frontier.pop() # priority = ForwCost(s)
  if IsGoal(s): return s # Found goal
  explored.add(s)
  for a in Actions(s):
    t = Succ(s, a)
    if t in explored: continue
    frontier.update(t, priority + Cost(s, a))
```


Analysis of uniform cost search

Notes

Proposition: Correctness
When a state s is popped off the frontier, `priority(s)` is the true forward cost $\text{ForwCost}(s)$. Therefore, UCS terminates with the optimal path.

Proof:



Implementation

Priority queue (UCS):

- Pop and update operations take $O(\log(\text{number of states on frontier}))$ time

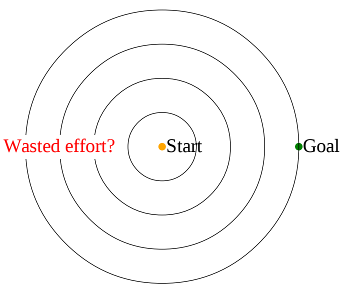
Regular queue (BFS):

- Pop and update Operations take $O(1)$ time
- Works only when edge costs are all equal

Outline

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - **A* search**
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

Can uniform cost search be improved?



Desiderata: prioritize exploring states "probably closer" to the goal

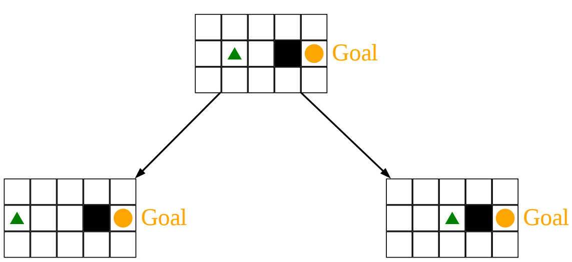
Heuristics

Definition: Heuristic function h

A heuristic $h(s)$ is any estimate of $\text{BackCost}(s)$, the minimum cost from s to a goal.

Example: $h(s) = \text{Distance}(\text{Location}(s), \text{GoalLocation})$

Effect of heuristic



$\text{ForwCost}(s) = 1, h(s) = 4$ $\text{ForwCost}(s) = 1, h(s) = 2$

Point: two actions result in same $\text{ForwCost}(s)$, but $h(s)$ breaks the tie

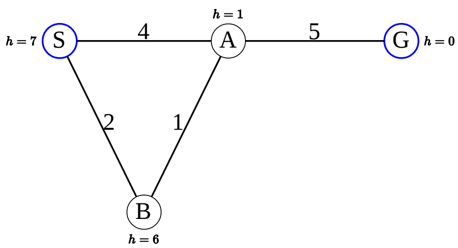
A* algorithm

Algorithm: A* search

```
explored = set()
frontier = PriorityQueue()
frontier.update(initState, h(initState))
while True:
  if frontier.size() == 0: return None
  s, priority = frontier.pop() # priority = ForwCost(s) + h(s)
  if IsGoal(s): return s # Found goal
  explored.add(s)
  for a in Actions(s):
    t = Succ(s, a)
    if t in explored: continue
    frontier.update(t, priority + Cost(s, a) + h(t) - h(s))
```

Does A* always work?

Notes



No.

Conditions on heuristic function

For A* to work, need conditions on the heuristic.

Definition: Admissibility
A heuristic h is admissible if $0 \leq h(s) \leq \text{BackCost}(s)$.

Definition: Consistency
A heuristic h is consistent if

$h(s) \leq \text{Cost}(s,a) + h(\text{Succ}(s,a))$, and

$h(s) = 0$ for all goal states s.

Consistency implies admissibility.

Analysis of A*

Proposition: Correctness
If h is consistent, A* returns the minimum cost path.

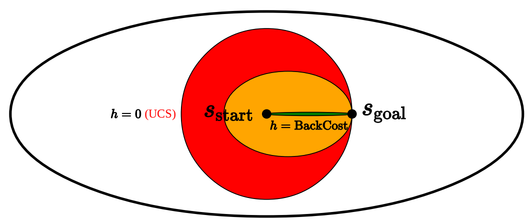
Proof:

- Define $\text{Cost}_h(s,a) \stackrel{\text{def}}{=} \text{Cost}(s,a) + [h(\text{Succ}(s,a)) - h(s)]$.
- Running A* on Cost is equivalent to UCS on Cost_h.
- By consistency, $\text{Cost}_h(s,a) \geq 0$, so UCS on Cost_h returns a path with minimum PathCost_h.
- Since $\text{PathCost}_h(p) = \text{PathCost}(p) - h(s_{\text{start}})$, running UCS on Cost_h is equivalent to UCS on Cost.

Analysis of A*

Proposition: Speed
A* explores only states s with

$$\text{ForwCost}(s) + h(s) \leq \min_p \text{PathCost}(p).$$

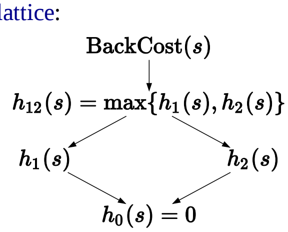


Comparing heuristics

Proposition: Dominance
A $h(s)$ dominates (is better than) $h'(s)$ if:

For all s, $h(s) \geq h'(s)$.

Heuristics form a lattice:

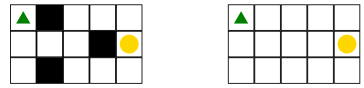


Outline

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - **Heuristics via relaxation**
    - Automatically deriving heuristics
  - Bellman-Ford for negative costs

How do we get good heuristics? Just relax...

General idea: analytic solutions



Hard Easy

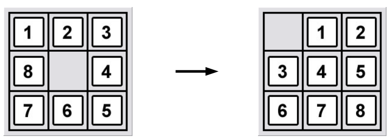
Remove constraints / add edges with cost 1 (e.g., $(1,1)$ to $(2,1)$)

Resulting heuristic has closed form:

$$h(s) = \text{Distance}(\text{Location}(s), \text{GoalLocation})$$

Lesson: try to make problem solvable without search

General idea: independent subproblems



Original problem: tiles cannot overlap (constraint)

Relaxed problem: tiles can overlap (no constraint)

Lesson: decompose problem into independent subproblems

General idea: state abstraction

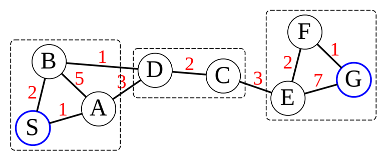
Task: go from home to office, hitting $p_1, \dots, p_k$ on the way

State: $(p, b_1, \dots, b_k)$ where p is position and b_i is whether hit p_i

Relaxation: only keep track of (p, b_1, b_2), not $b_3, \dots, b_k$

Effect: treat ((5,2), 1, 0, 1, 0) and ((5,2), 1, 0, 0, 0) the same

Lesson: collapse similar states into one abstract state



General idea: state abstraction

Definition: Abstraction function

An abstraction α maps a (concrete) state s to an abstract state c (thereby defining a partitioning of the states).

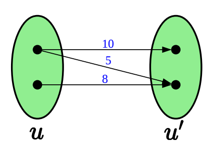
Examples:

- Take sign: $\alpha(3) = +$, $\alpha(-4) = -$
- Drop attributes: $\alpha(\{x : 3, y : -4, d : \text{East}\}) = \{x : 3, y : -4\}$

General idea: state abstraction

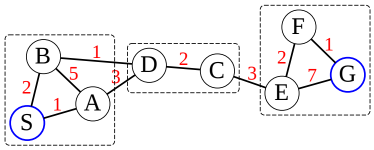
Definition: Abstract model

- $\text{States}^\alpha = \{\alpha(s) : s \in \text{States}\}$
- $\text{IsGoal}^\alpha(u) = [\text{IsGoal}(s) \text{ for some } s : \alpha(s) = u]$
- $\text{Actions}^\alpha, \text{Succ}^\alpha, \text{Cost}^\alpha$: cost of edge from u to u' is minimum over cost of edges from $s \in \alpha^{-1}(u)$ to $s' \in \alpha^{-1}(u')$



(abstract cost from u to u' is 5)

Heuristic function based on abstraction



Heuristic: define $h(s)$ to be $\text{BackCost}^{\alpha}(\alpha(s))$, minimum cost from $\alpha(s)$ to an abstract goal (in the abstract graph).

Types of relaxation

- Analytic solutions: same state space, but solve in closed form
- State abstraction: reduce state space, use search
- Independent subproblems: break problem into several smaller ones

What's common to the above?

Unifying principle: relaxation

Definition: Relaxed model
A relaxed model is a one with lower costs:
$\text{Cost}'(s,a) \le \text{Cost}(s,a)$.

Heuristic: Define $h(s) = \text{BackCost}'(s)$, the minimum cost from s to a goal state using $\text{Cost}'(s,a)$.

Consistency of $h(s)$:

$$h(s) \le \text{Cost}'(s,a) + h(\text{Succ}(s,a)) \text{ [by triangle inequality]}$$
$$\le \text{Cost}(s,a) + h(\text{Succ}(s,a)) \text{ [by relaxation]}$$

Point: relaxed model is only useful if **easier** to solve

Outline

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - **Automatically deriving heuristics**
  - Bellman-Ford for negative costs

Motivation

Task: start at Home, go visit Office and Store and come back

Deterministic state space model

```
initState = ["Home", set()]
def IsGoal(s):
  return s[0] == "Home" and s[1] == set(["Office", "Store"])
def Actions(s): return ["Visit", ...]
def Succ(s, a): ...
def Cost(s, a): return 1
```

\+ search algorithm

Problem:

- Search algorithms treat states as black boxes
- Can't exploit **structure** of task to generate A* heuristics

Notes

Peering inside a state space model

Represent state as a set of **fluents**; actions add/delete fluents.

PDDL instance

```
Init: At(Home)
Goal: At(Home), Visited(Office), Visited(Store)
Action: Move(p,q) # for all p, q
  Precond: At(p), Adjacent(p,q)
  Effect: At(q), -At(p)
Action: Visit(p) # for all p
  Precond: At(p)
  Effect: Visited(p)
```

↓

state space model + search algorithms

Relaxations for getting heuristics

PDDL instance

```
Init: At(Home)
Goal: At(Home), Visited(Office), Visited(Store)
Action: Move(p,q) # for all p, q
  Precond: At(p), Adjacent(p,q)
  Effect: At(q), -At(p)
Action: Visit(p) # for all p
  Precond: At(p)
  Effect: Visited(p)
```

Relaxations:

- Remove a goal condition (e.g., remove Visited(Office))
- Remove an action precondition (e.g., remove Adjacent(p,q))
- Remove all instances of a fluent (e.g., remove At(p))
- Remove all instances of a negative term (e.g., remove -At(p))

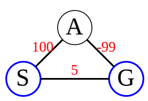
Outline

- Definition of deterministic state space models
- Motivating applications
- Some challenges: uncertainty and continuous spaces
- Algorithms
  - DAG search, DFS
  - Uniform cost search, BFS
  - A* search
    - Heuristics via relaxation
    - Automatically deriving heuristics
  - **Bellman-Ford for negative costs**

Cycles, negative costs

Which would you choose (utility is maximize money):

- Option 1: pay $\$5$
- Option 2: pay $\$100$, get $\$99$ refund later



Problem with current algorithms:

- DAG search: infinite loop (cyclic graph)
- UCS/A*: choose $\$5$ path

Cycles, negative costs

Notes

$$\text{BackCost}(s) = \begin{cases} 0 & \text{if } s = s_{\text{goal}} \\ \min_{a \in \text{Actions}(s)} [\text{Cost}(s,a) + \text{BackCost}(\text{Succ}(s,a))] & \text{otherwise.} \end{cases}$$

Algorithm: Bellman-Ford algorithm

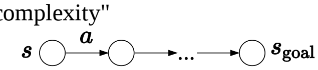
```
backCost = {}
for s in States: backCost[s] = float('inf')
backCost[goalState] = 0
for _ in range(len(States)): # Repeat |States| times
  for s in States: # For each s, update backCost[s]
    for a in Actions(s):
      t = Succ(s, a)
      backCost[s] = min(backCost[s], Cost(s, a) + backCost[t])
```

Key property: After i iterations, BackCost is correct for minimum cost paths to the goal state with at most i edges.

Summary of algorithms

$$\text{BackCost}(s) = \begin{cases} 0 & \text{if } s = s_{\text{goal}} \\ \min_{a \in \text{Actions}(s)} [\text{Cost}(s,a) + \text{BackCost}(\text{Succ}(s,a))] & \text{otherwise.} \end{cases}$$

Unifying idea: construct minimum cost paths from s to the goal state in order of "complexity"



- DAG search: relies on topological ordering of states (possible due to acyclicity)
- Uniform cost search (reversed): orders by path cost (possible due to non-negative costs)
- Bellman-Ford: orders by number of edges (no structure)

Next time: non-deterministic state space models...