Search II

Roadmap

Modeling

Modeling Search Problems

Algorithms

Tree Search

Dynamic Programming

Uniform Cost Search

Programming and Correctness of UCS

A*

A* Relaxations

Learning

Structured Perceptron

Key idea: state

A **state** is a summary of all the past actions sufficient to choose future actions **optimally**.

past actions (all cities)	1 3 4 6 5 3
state (current city)	1 3 4 6 5 3

Review

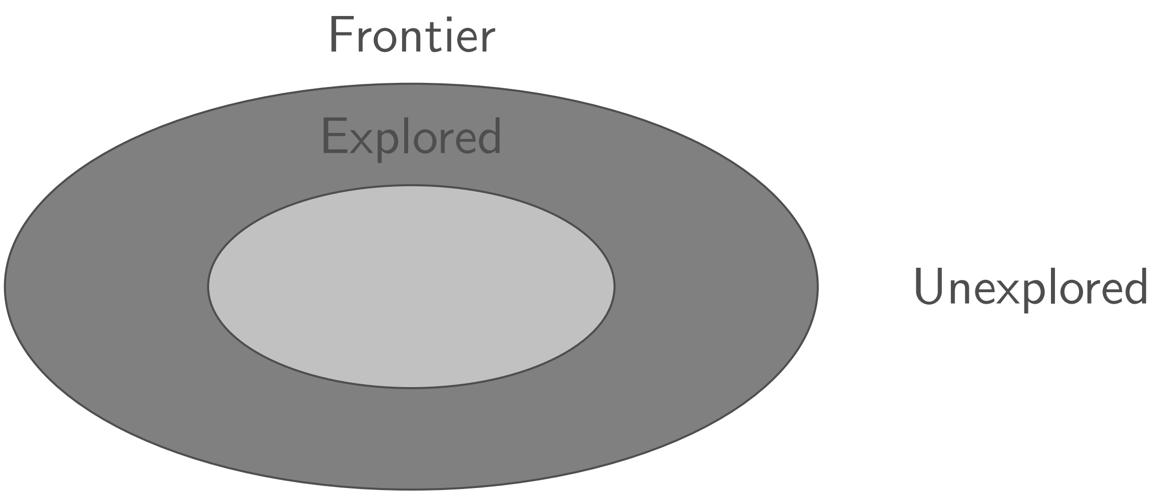
Definition: search problem

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

Objective: find the minimum cost path from s_{start} to an s satisfying IsEnd(s).

Search: uniform cost search correctness

High-level strategy



- Explored: states we've found the optimal path to
- Frontier: states we've seen, still figuring out how to get there cheaply
- Unexplored: states we haven't seen

Uniform cost search (UCS)

Algorithm: uniform cost search [Dijkstra, 1956]

Add s_{start} to **frontier** (priority queue)

Repeat until frontier is empty:

  Remove s with smallest priority p from frontier

  If IsEnd(s): return solution

  Add s to **explored**

  For each action $a \in$ Actions(s):

    Get successor $s' \leftarrow$ Succ(s, a)

    If s' already in explored: continue

    Update **frontier** with s' and priority $p +$ Cost(s, a)

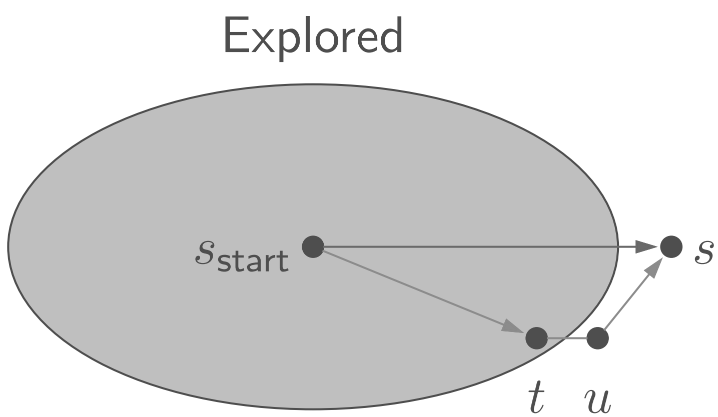
[live solution: `Uniform Cost Search`]

Analysis of uniform cost search

Theorem: correctness

When a state s is popped from the frontier and moved to explored, its priority is PastCost(s), the minimum cost to s.

Proof:



DP versus UCS

N total states, n of which are closer than end state

Algorithm	**Cycles?**	**Action costs**	**Time/space**
DP	no	any	$O(N)$
UCS	yes	≥ 0	$O(n \log n)$

Note: UCS potentially explores fewer states, but requires more overhead to maintain the priority queue

Note: assume number of actions per state is constant (independent of n and N)

Summary

- Tree search: memory efficient, suitable for huge state spaces but exponential worst-case running time

- State: summary of past actions sufficient to choose future actions optimally

- Graph search: dynamic programming and uniform cost search construct optimal paths (exponential savings!)

- Next: searching faster with A*, learning action costs (if time)

think and share

Question

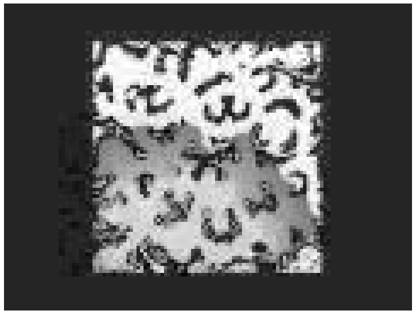
Suppose we want to travel from city 1 to city n (going only forward) and back to city 1 (only going backward). It costs $c_{ij} \geq 0$ to go from i to j. Which of the following algorithms can be used to find the minimum cost path (select all that apply)?

- depth-first search
- breadth-first search
- dynamic programming
- uniform cost search

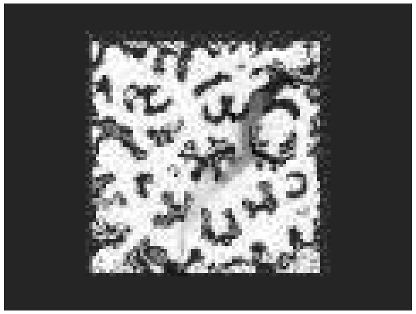
Search: A*

A* algorithm

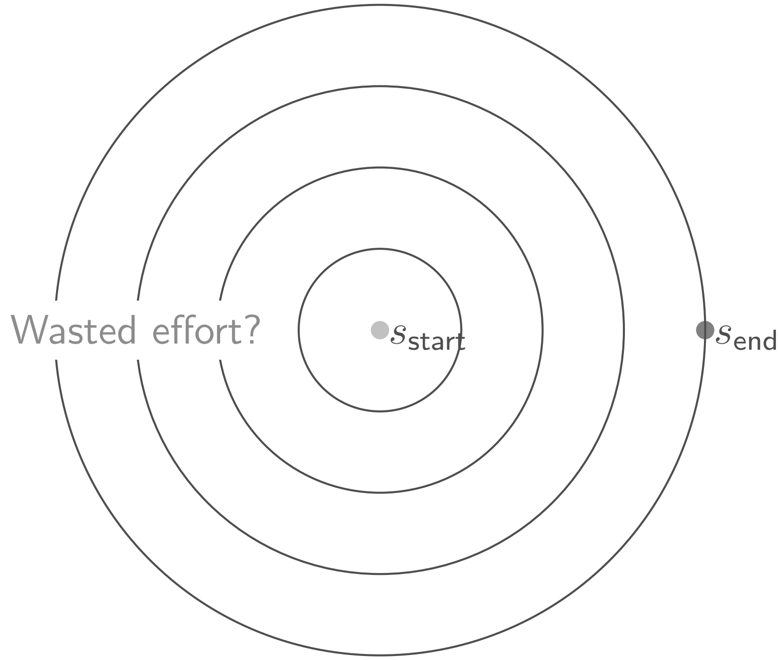
UCS in action:



A* in action:



Can uniform cost search be improved?



Problem: UCS orders states by cost from s_{start} to s

Goal: take into account cost from s to s_{end}

Exploring states

UCS: explore states in order of $\text{PastCost}(s)$

$$s_{\text{start}} \xrightarrow[\text{PastCost}(s)]{} s \xrightarrow[\text{FutureCost}(s)]{} s_{\text{end}}$$

Ideal: explore in order of $\text{PastCost}(s) + \text{FutureCost}(s)$

A*: explore in order of $\text{PastCost}(s) + h(s)$

Definition: Heuristic function

A heuristic $h(s)$ is any estimate of $\text{FutureCost}(s)$.

A* search

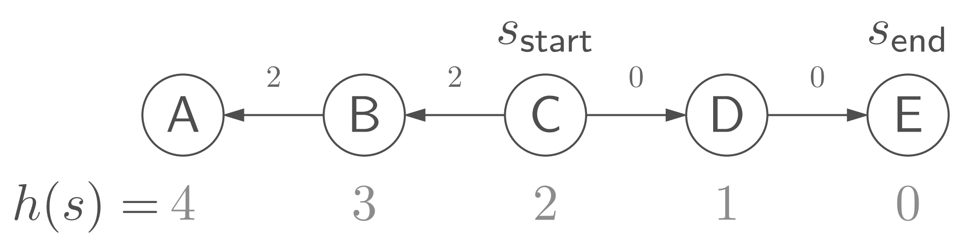
Algorithm: A* search [Hart/Nilsson/Raphael, 1968]

Run uniform cost search with **modified edge costs**:

$$\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)$$

Intuition: add a penalty for how much action a takes us away from the end state

Example:



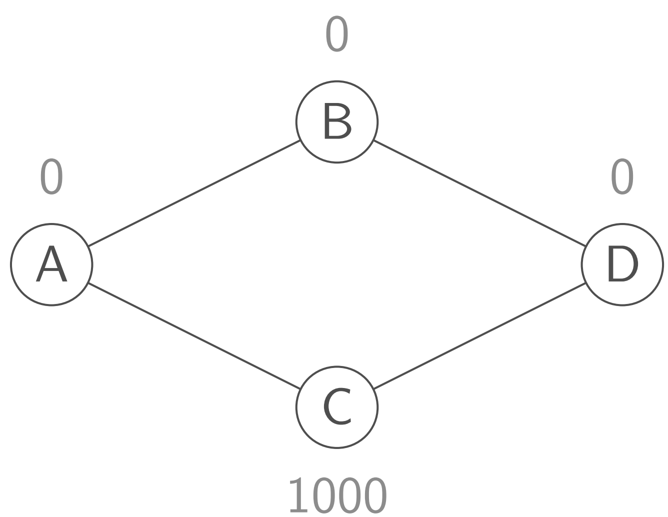
$\text{Cost}'(C,B) = \text{Cost}(C,B) + h(B) - h(C) = 1 + (3-2) = 2$

An example heuristic

Will any heuristic work?

No.

Counterexample:



Doesn't work because of **negative modified edge costs**!

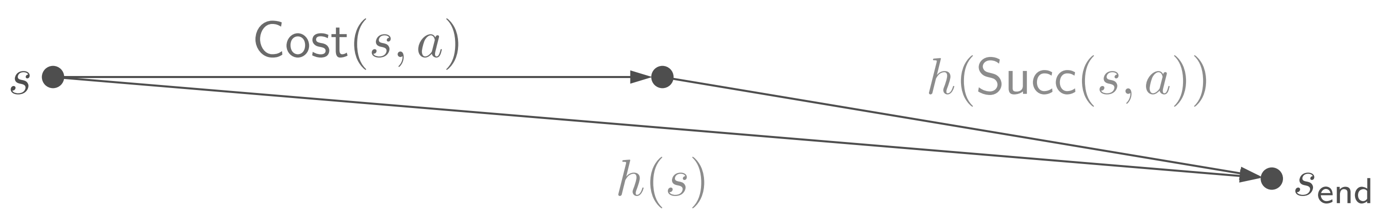
Consistent heuristics

Definition: consistency

A heuristic h is **consistent** if

- $\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0$.

Condition 1: needed for UCS to work (triangle inequality).



Condition 2: $\text{FutureCost}(s_{\text{end}}) = 0$ so match it.

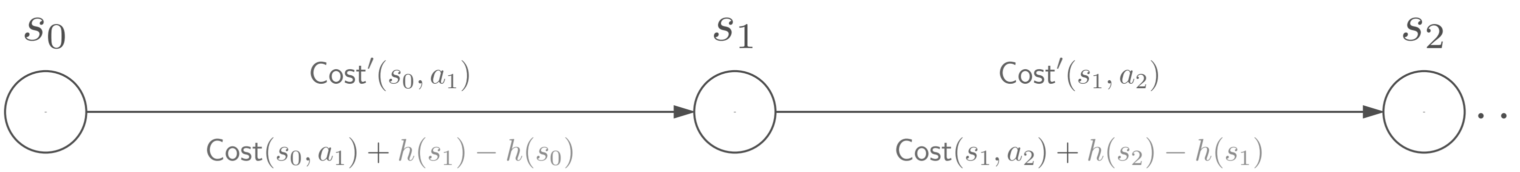
Correctness of A*

Proposition: correctness

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

Proof of A* correctness

- Consider any path $[s_0, a_1, s_1, \ldots, a_L, s_L]$:



- Key identity:

$$\underbrace{\sum_{i=1}^{L} \text{Cost}'(s_{i-1}, a_i)}_{\text{modified path cost}} = \underbrace{\sum_{i=1}^{L} \text{Cost}(s_{i-1}, a_i)}_{\text{original path cost}} + \underbrace{h(s_L) - h(s_0)}_{\text{constant}}$$

- Therefore, A* (finding the minimum cost path using modified costs) solves the original problem (even though edge costs are all different!)

Efficiency of A*

Theorem: efficiency of A*

A* explores all states s satisfying
$\text{PastCost}(s) \le \text{PastCost}(s_{\text{end}}) - h(s)$

Interpretation: the larger $h(s)$, the better

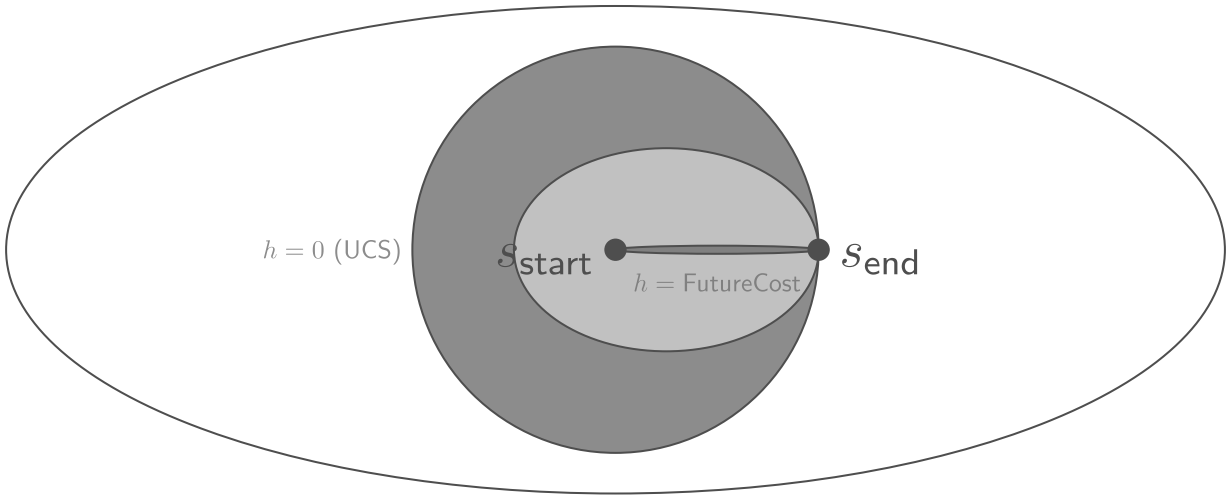
Proof: A* explores all s such that

$$\text{PastCost}(s) + h(s)$$

$$\le$$

$$\text{PastCost}(s_{\text{end}})$$

Amount explored

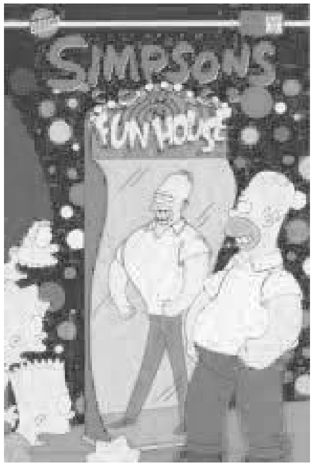


- If $h(s) = 0$, then A* is same as UCS.
- If $h(s) = \text{FutureCost}(s)$, then A* only explores nodes on a minimum cost path.
- Usually $h(s)$ is somewhere in between.

A* search

Key idea: distortion

A* distorts edge costs to favor end states.



Admissibility

Definition: admissibility

A heuristic $h(s)$ is admissible if
$$h(s) \leq \text{FutureCost}(s)$$

Intuition: admissible heuristics are optimistic

Theorem: consistency implies admissibility

If a heuristic $h(s)$ is **consistent**, then $h(s)$ is **admissible**.

Proof: use induction on $\text{FutureCost}(s)$

Search: A* relaxations

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

Relaxation

Intuition: ideally, use $h(s) = \text{FutureCost}(s)$, but that's as hard as solving the original problem.

Key idea: relaxation

Constraints make life hard. Get rid of them.

But this is just for the heuristic!

Relaxation overview

reduce costs

remove constraints

closed form solution

easier search

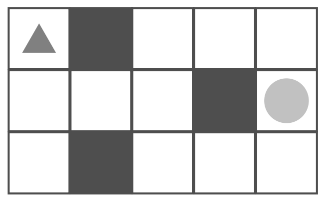
independent subproblems

combine heuristics using max

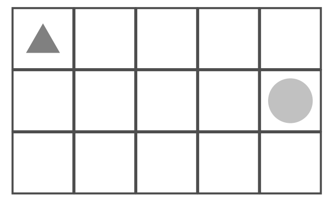
Closed form solution

Example: knock down walls

Goal: move from triangle to circle



Hard



Easy

Heuristic:

$$h(s) = \text{ManhattanDistance}(s, (2,5))$$

e.g., $h((1,1)) = 5$

Easier search

Example: original problem

Start state: 1

Walk action: from s to $s+1$ (cost: 1)

Tram action: from s to $2s$ (cost: 2)

End state: n

Constraint: can't have more tram actions than walk actions.

State: (location, #**walk** - #**tram**)

Number of states goes from $O(n)$ to $O(n^2)$!

Easier search

Example: relaxed problem

Start state: 1

Walk action: from s to $s+1$ (cost: 1)

Tram action: from s to $2s$ (cost: 2)

End state: n

~~Constraint: can't have more tram actions than walk actions.~~

Original state: (location, **#walk** - **#tram**)

Relaxed state: location

Easier search

- Compute relaxed $\text{FutureCost}_{\text{rel}}(\text{location})$ for **each** location $(1, \ldots, n)$ using dynamic programming or UCS

Example: reversed relaxed problem

Start state: n

Walk action: from s to $s-1$ (cost: 1)

Tram action: from s to $s/2$ (cost: 2)

End state: 1

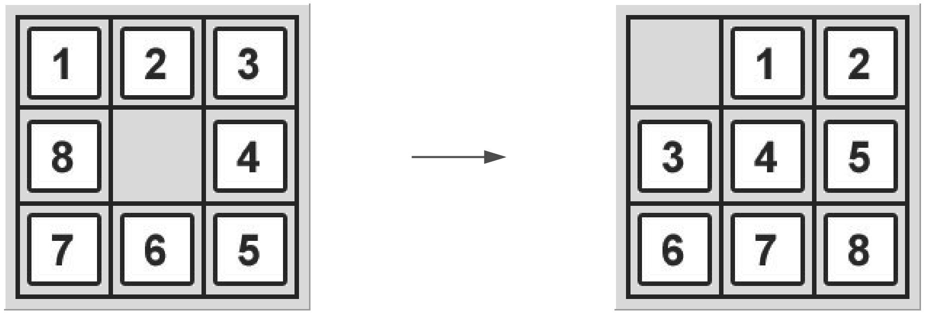
Modify UCS to compute all past costs in reversed relaxed problem
(equivalent to future costs in relaxed problem!)

- Define heuristic for original problem:

$$h((\text{location}, \#\text{walk-}\#\text{tram})) = \text{FutureCost}_{\text{rel}}(\text{location})$$

Independent subproblems

[8 puzzle]



Original problem: tiles cannot overlap (constraint)

Relaxed problem: tiles can overlap (no constraint)

Relaxed solution: 8 indep. problems, each in closed form

Key idea: independence

Relax original problem into independent subproblems.

General framework

Removing constraints

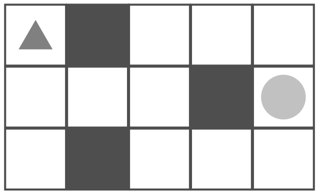
(knock down walls, walk/tram freely, overlap pieces)

$\updownarrow$

Reducing edge costs

(from ∞ to some finite cost)

Example:



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

Relaxed: $\text{Cost}_{\text{rel}}((1,1), \text{East}) = 1$

General framework

Definition: relaxed search problem

A **relaxation** P_{rel} of a search problem P has costs that satisfy:

$$\text{Cost}_{\text{rel}}(s, a) \leq \text{Cost}(s, a).$$

Definition: relaxed heuristic

Given a relaxed search problem P_{rel}, define the **relaxed heuristic** $h(s) = \text{FutureCost}_{\text{rel}}(s)$, the minimum cost from s to an end state using $\text{Cost}_{\text{rel}}(s, a)$.

General framework

Theorem: consistency of relaxed heuristics

Suppose $h(s) = \text{FutureCost}_{\text{rel}}(s)$ for some relaxed problem P_{rel}.

Then $h(s)$ is a consistent heuristic.

Proof:

$$
\begin{aligned}
h(s) &\le \text{Cost}_{\text{rel}}(s,a) + h(\text{Succ}(s,a)) \text{ [triangle inequality]} \\
&\le \text{Cost}(s,a) + h(\text{Succ}(s,a)) \text{ [relaxation]}
\end{aligned}
$$

Tradeoff

Efficiency:

$h(s) = \text{FutureCost}_{\text{rel}}(s)$ must be easy to compute

Closed form, easier search, independent subproblems

Tightness:

heuristic $h(s)$ should be close to $\text{FutureCost}(s)$

Don't remove too many constraints

Max of two heuristics

How do we combine two heuristics?

Proposition: max heuristic

Suppose $h_1(s)$ and $h_2(s)$ are consistent.
Then $h(s) = \max\{h_1(s), h_2(s)\}$ is consistent.

Proof: exercise

Search: recap

Modeling: 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?

Inference

Algorithms

Tree Search

Dynamic Programming

Uniform Cost Search

Programming and Correctness of UCS

A*

A* Relaxations

Dynamic programming

Key idea: state

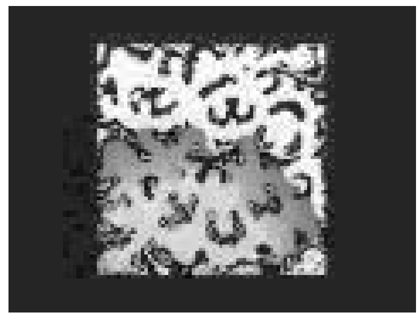
A **state** is a summary of all the past actions sufficient to choose future actions **optimally**.

past actions (all cities)	1 3 4 6
state (current city)	1 3 4 6

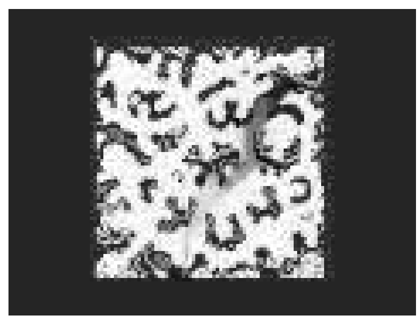
A* algorithm

Add in heuristic estimate of future costs.

UCS in action:



A* in action:



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

Relaxation (breaking the rules)

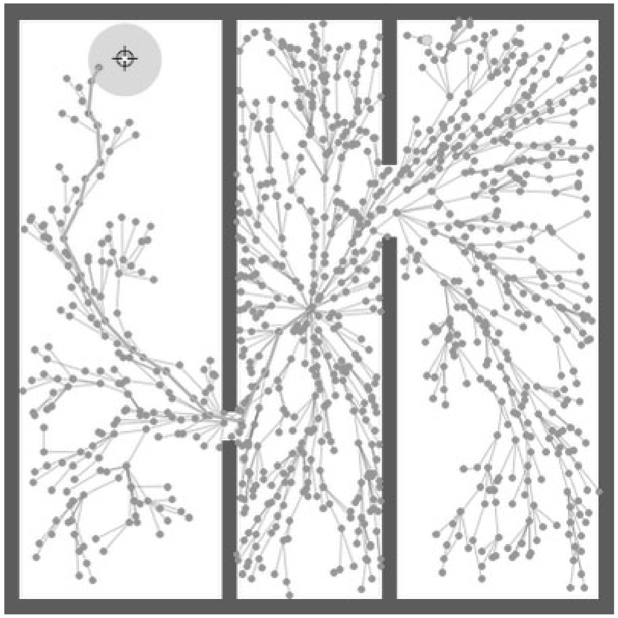
A framework for producing consistent heuristics.

Key idea: relaxation

Constraints make life hard. Get rid of them.

But this is just for the heuristic!

Outlook: Sampling Based Planning Algorithms



Probabilistic Roadmaps (PRM) and Rapidly exploring Random Trees (RRT)

Next time: MDPs

When actions have unknown consequences...

Search: structured perceptron [optional]

Search

Transportation example

Start state: 1

Walk action: from s to $s+1$ (cost: 1)

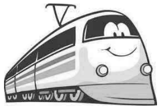
Tram action: from s to $2s$ (cost: 2)

End state: n

search algorithm

walk walk tram tram tram walk tram tram

(minimum cost path)



Learning

Transportation example

Start state: 1

Walk action: from s to $s+1$ (cost: **?**)

Tram action: from s to $2s$ (cost: **?**)

End state: n

walk walk tram tram tram walk tram tram

↓ learning algorithm

walk cost: **1**, tram cost: **2**

Learning as an inverse problem

Forward problem (search):

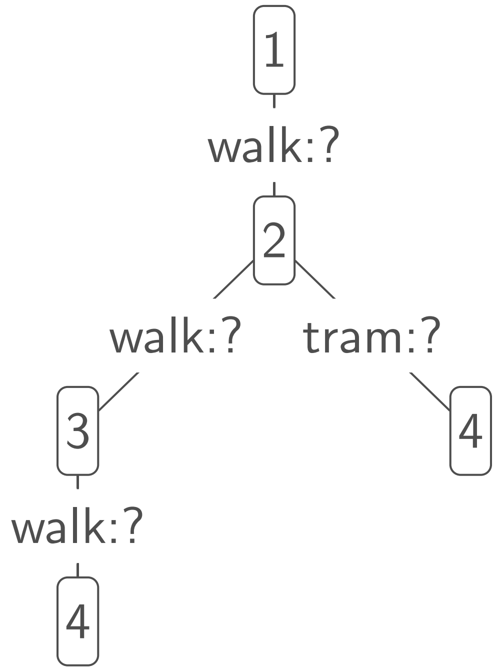
$$\text{Cost}(s,a) \longrightarrow (a_1,\dots,a_k)$$

Inverse problem (learning):

$$(a_1,\dots,a_k) \longrightarrow \text{Cost}(s,a)$$

Prediction (inference) problem

Input x: search problem without costs



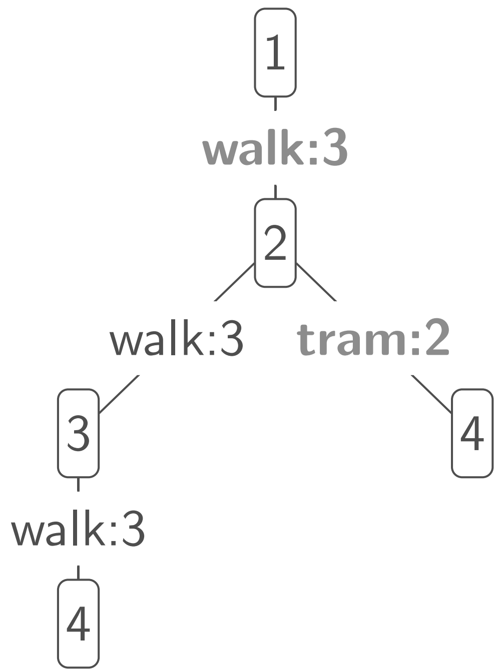
Output y: solution path

walk walk walk

Tweaking costs

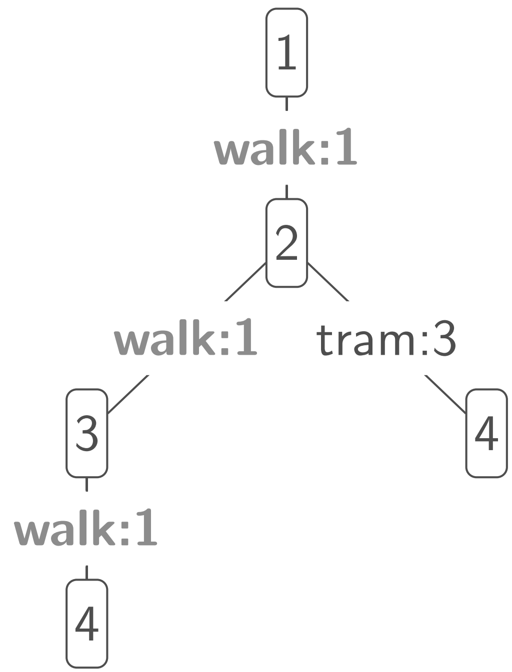
Costs: {walk:3, tram:2}

Minimum cost path:



Costs: {walk:1, tram:3}

Minimum cost path:



Modeling costs (simplified)

Assume costs depend only on the action:

$$\text{Cost}(s, a) = \mathbf{w}[a]$$

Candidate output path:

$$y\text{:} \quad \boxed{s_0} \xrightarrow{a_1 : \mathbf{w}[a_1]} \boxed{s_1} \xrightarrow{a_2 : \mathbf{w}[a_2]} \boxed{s_2} \xrightarrow{a_3 : \mathbf{w}[a_3]} \boxed{s_3}$$

Path cost:

$$\text{Cost}(y) = \mathbf{w}[a_1] + \mathbf{w}[a_2] + \mathbf{w}[a_3]$$

Learning algorithm

Algorithm: Structured Perceptron (simplified)

- For each action: $\mathbf{w}[a] \leftarrow 0$
- For each iteration $t = 1, \dots T$:
  - For each training example $(x, y) \in \mathcal{D}_{\text{train}}$:
    - Compute the minimum cost path y' given $\mathbf{w}$
    - For each action $a \in y$: $\mathbf{w}[a] \leftarrow \mathbf{w}[a] - 1$
    - For each action $a \in y'$: $\mathbf{w}[a] \leftarrow \mathbf{w}[a] + 1$

- Try to decrease cost of true y (from training data)
- Try to increase cost of predicted y' (from search)

[live solution: `Structured Perceptron`]

Generalization to features (skip)

Costs are parametrized by feature vector:

$$\text{Cost}(s, a) = \mathbf{w} \cdot \phi(s, a)$$

Example:

$$y: \quad \boxed{s_0} \xrightarrow{a_1 : \mathbf{w} \cdot [1, 0, 1]} \boxed{s_1} \xrightarrow{a_2 : \mathbf{w} \cdot [1, 2, 0]} \boxed{s_2}$$

$$\mathbf{w} = [3, -1, -1]$$

Path cost:

$\text{Cost}(y) = 2 + 1 = 3$

Learning algorithm (skip)

Algorithm: Structured Perceptron [Collins, 2002]

- For each action: $\mathbf{w} \leftarrow 0$
- For each iteration $t = 1, \dots T$:
  - For each training example $(x, y) \in \mathcal{D}_{\text{train}}$:
    - Compute the minimum cost path y' given $\mathbf{w}$
    - $\mathbf{w} \leftarrow \mathbf{w} - \phi(y) + \phi(y')$

- Try to decrease cost of true y (from training data)
- Try to increase cost of predicted y' (from search)

Applications

- Part-of-speech tagging

Fruit flies like a banana. $\longrightarrow$ Noun Noun Verb Det Noun

- Machine translation

la maison bleue $\longrightarrow$ *the blue house*

Homework

due: next week

作业 3-周3-torch-nn