Search II:
Learning parameters, A*

Hwanjo Yu

POSTECH

http://di.postech.ac.kr/hwanjoyu

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

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

$\Big\downarrow$ learning algorithm

walk cost: 1, tram cost: 2

Learning as an inverse problem

Forward problem (search):

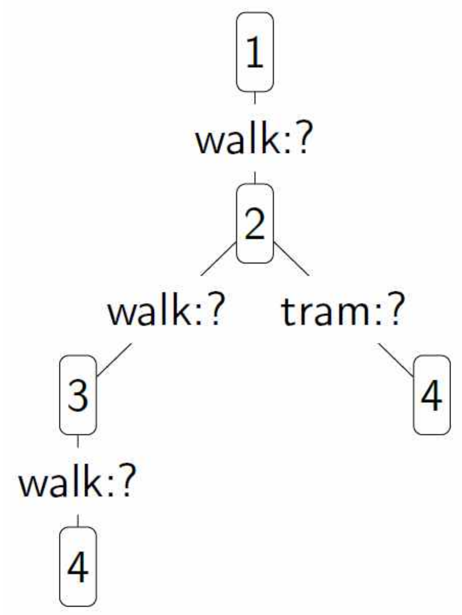
$$\mathrm{Cost}(s, a) \rightarrow (a_1, \dots, a_k)$$

Inverse problem (learning):

$$(a_1, \dots, a_k) \rightarrow \mathrm{Cost}(s, a)$$

Prediction (inference) problem

Input x: search problem without costs



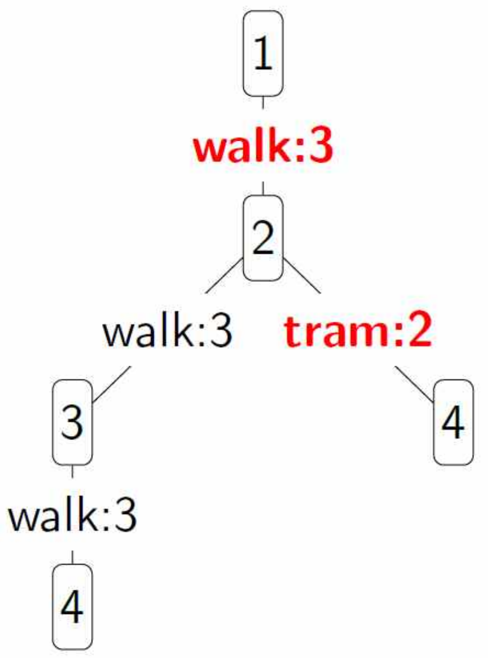
Output y: solution path

walk walk walk

Tweaking costs

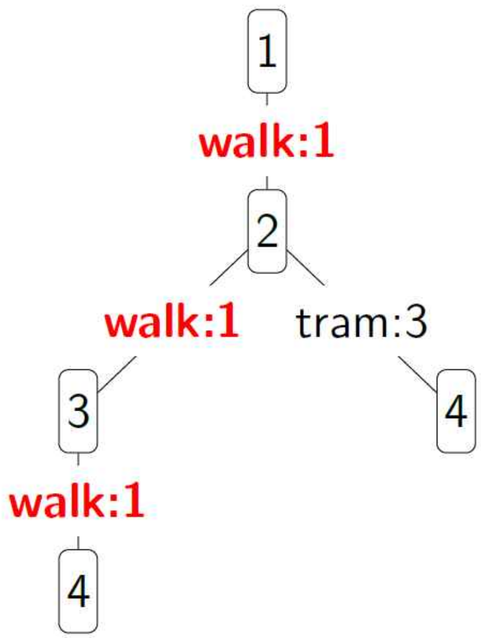
Costs: {walk:3, tram:2}

Minimum cost path:



Costs: {walk:1, tram:3}

Minimum cost path:

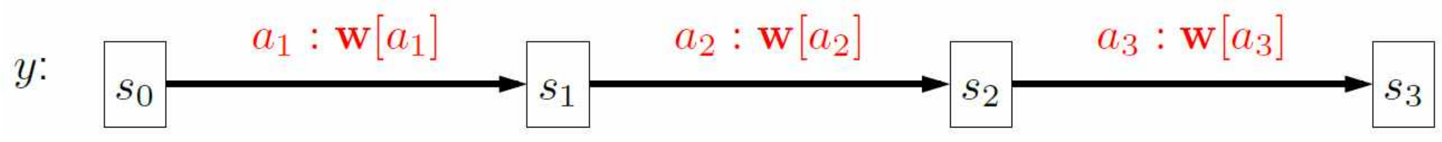


Modeling costs (simplified)

Assume costs depend only on the action:

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

Candidate output path:



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 $t = 1, \ldots T$:
  - For each training example $(x, y) \in D_{\text{train}}$:
    - 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)
- Note that if $y = y'$, there is no update.

- What is the implied objective (loss function)? (Notate $\mathbf{w}[a] \rightarrow \mathbf{w} \cdot \phi(a)$)

Generalization to features

Costs are parametrized by feature vector

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

Example:

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

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

Path cost:

$$\text{Cost}(y) = \mathbf{w} \cdot [1,0,1] + \mathbf{w} \cdot [1,2,0] = 2 + 1 = 3$$

Learning algorithm

Algorithm: Structured Perceptron

- For each action: $\mathbf{w} \leftarrow 0$
- For each iteration $t = 1, \ldots T$:
  - For each training example $(x, y) \in D_{\text{train}}$:
    - 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)

1. What is the implied objective (loss function)?
2. Can you modify it using hinge loss?
3. How the algorithm would change if using hinge loss?

Learning algorithm

Algorithm: Structured Perceptron

- For each action: $\mathbf{w} \leftarrow 0$
- For each iteration $t = 1, \ldots T$:
  - For each training example $(x, y) \in D_{\text{train}}$:
    - 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)

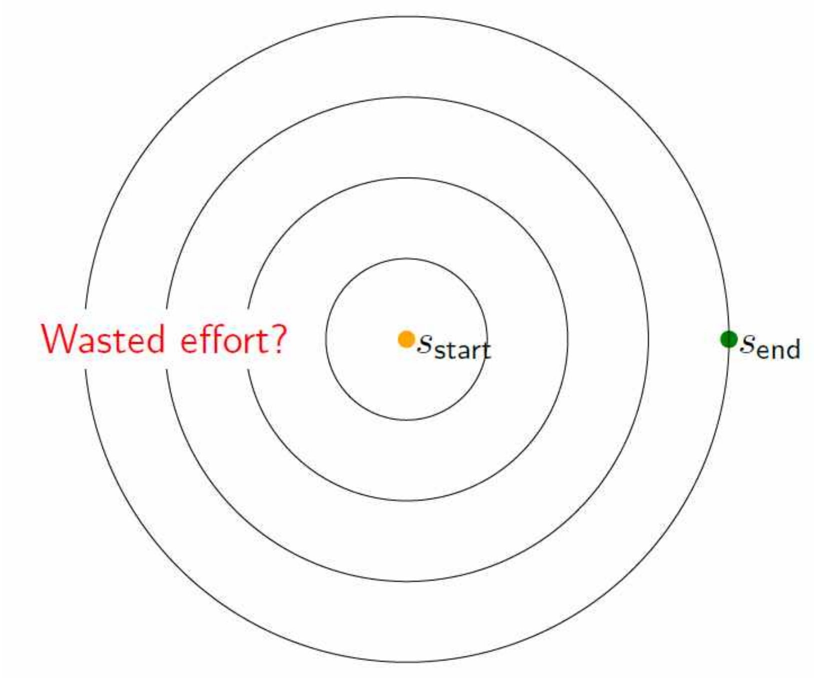
Roadmap

Learning costs

A* search

Relaxation

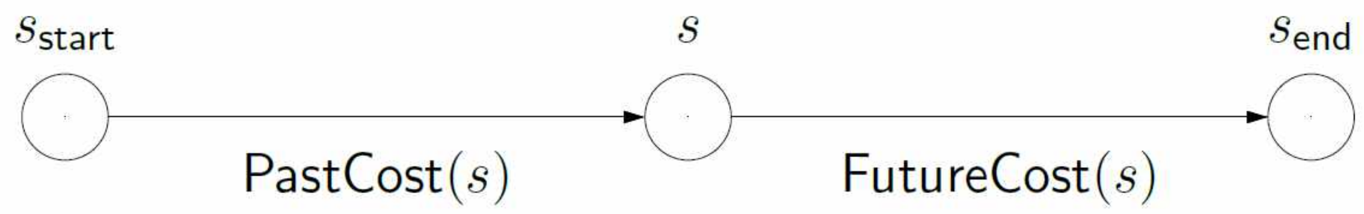
Can uniform cost search be improved?



- 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 PastCost(s)



- Ideal: explore in order of PastCost(s) + FutureCost(s)
- A*: explore in order of PastCost(s) + $h(s)$

Definition: Heuristic function

- A heuristic $h(s)$ is any estimate of FutureCost(s)

A* search

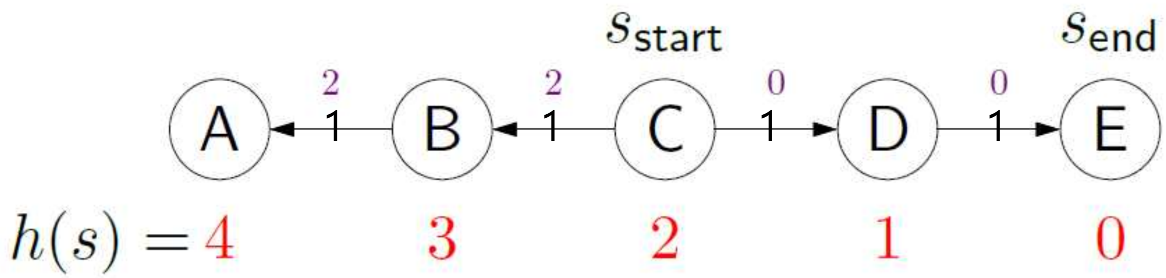
Algorithm: A* search [Hart/Nilsson/Raphael, 1968]

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

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

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

Example:



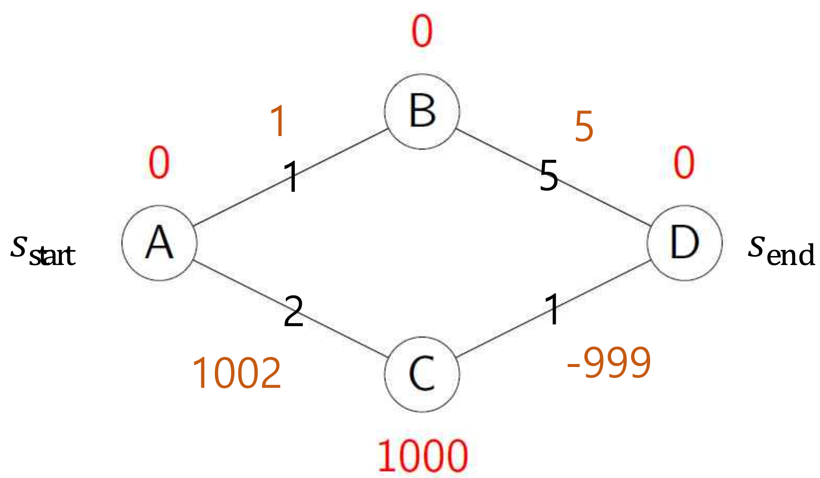
$$\text{Cost}'(C,B) = \text{Cost}(C,B) + h(B) - h(C) = 1 + (3-2) = 2$$

An example heuristic

Will any heuristic work?

No.

Counterexample:



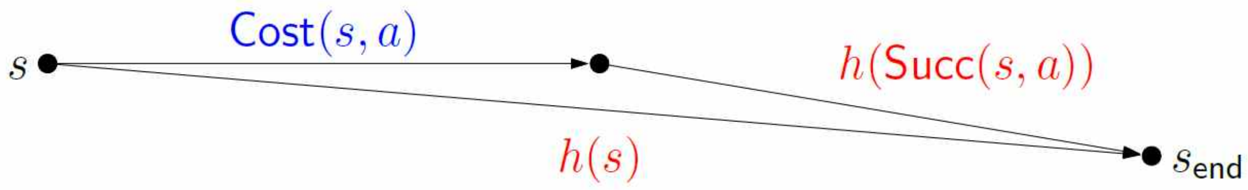
Doesn't work because of negative modified edge costs!

Consistent heuristics

Definition: consistency

- A heuristic h is **consistent** if
  - $\mathrm{Cost}'(s,a) = \mathrm{Cost}(s,a) + h(\mathrm{Succ}(s,a)) - h(s) \geq 0$
  - $h(s_{\mathrm{end}}) = 0$

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



- Condition 2: FutureCost(s_{end}) = 0, so match it.

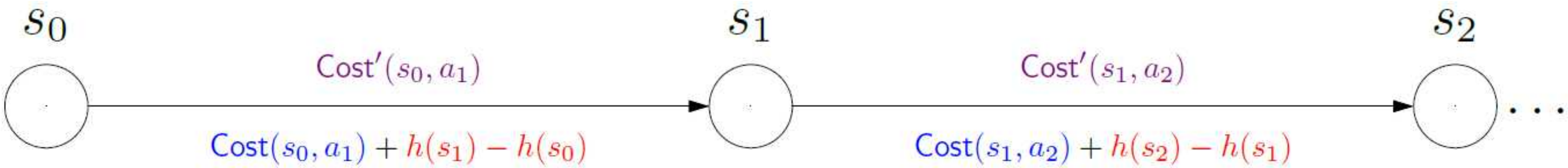
Correctness of A*

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

$$\text{PastCost}(s) \le \text{PastCost}(s_{\text{end}}) - h(s)$$

  - Note that UCS explores states in order of past cost, so that it explores every state whose past cost is less than the past cost of the end state.

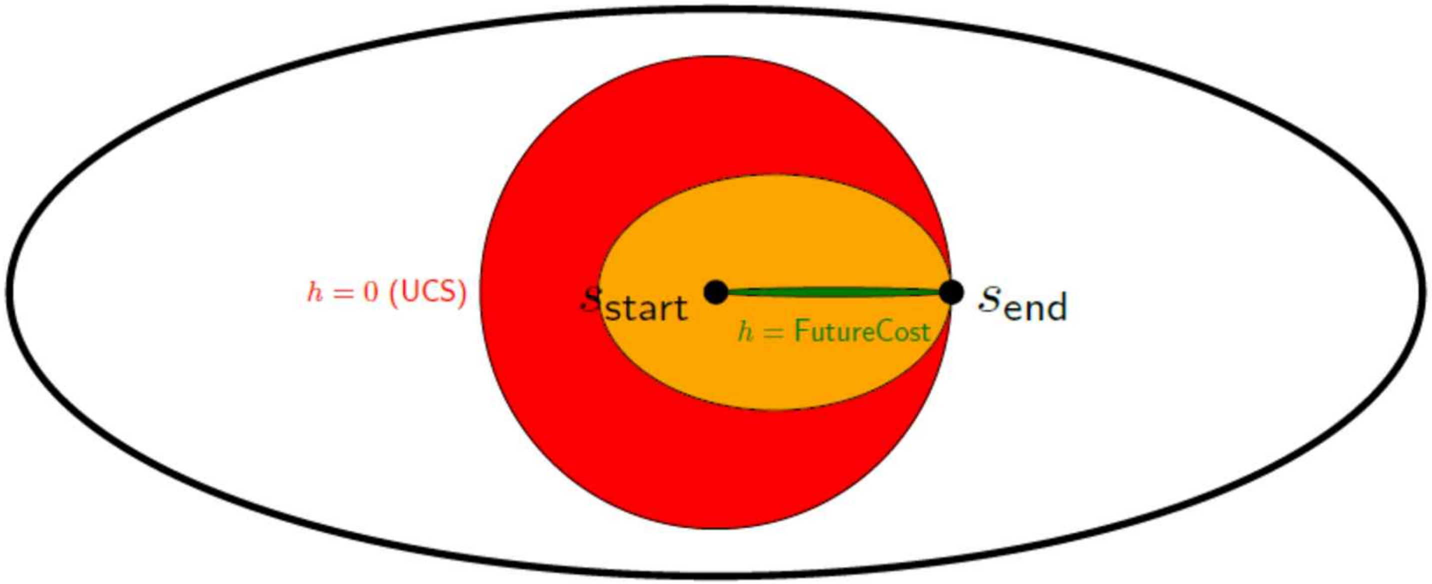
Proof: A* explores all s such that

$$\text{PastCost}(s) + h(s) - h(s_{\text{start}}) \le \text{PastCost}(s_{\text{end}}) + h(s_{\text{end}}) - h(s_{\text{start}})$$

$$\text{PastCost}(s) + h(s) \le \text{PastCost}(s_{\text{end}})$$

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

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 is $h(s)$ somewhere in between.

Admissibility

Definition: admissibility

- A heuristic $h(s)$ is **admissible** if $h(s) \leq \text{FutureCost}(s)$
- Note: A* explores all s such that $\text{PastCost}(s) + h(s) \leq \text{PastCost}(s_{\text{end}})$

Theorem: consistency implies admissibility

- If a heuristic $h(s)$ is **consistent**, then $h(s)$ is **admissible**
- Note: a consistent heuristic $h(s) \leq \text{Cost}(s, a) + h(s')$ and $h(s_{\text{end}}) = 0$

Proof:

If $h(s)$ is not admissible, i.e., $h(s) = \text{FutureCost}(s) + \alpha\ (\alpha > 0)$,

Then,

$\text{Cost}'(s', a) = \text{Cost}(s', a) + h(s_{\text{end}}) - h(s')$ for s' where $\text{Cost}(s', a) = \text{FutureCost}(s')$
$\text{Cost}'(s', a) = \text{FutureCost}(s') - (\text{FutureCost}(s') + \alpha) < 0$

Thus, $h(s)$ is not consistent

A* with admissible heuristic

1. If h is admissible, h is consistent?

2. If h is admissible, A* returns the minimum cost path?

Roadmap

Learning costs

A* search

Relaxation

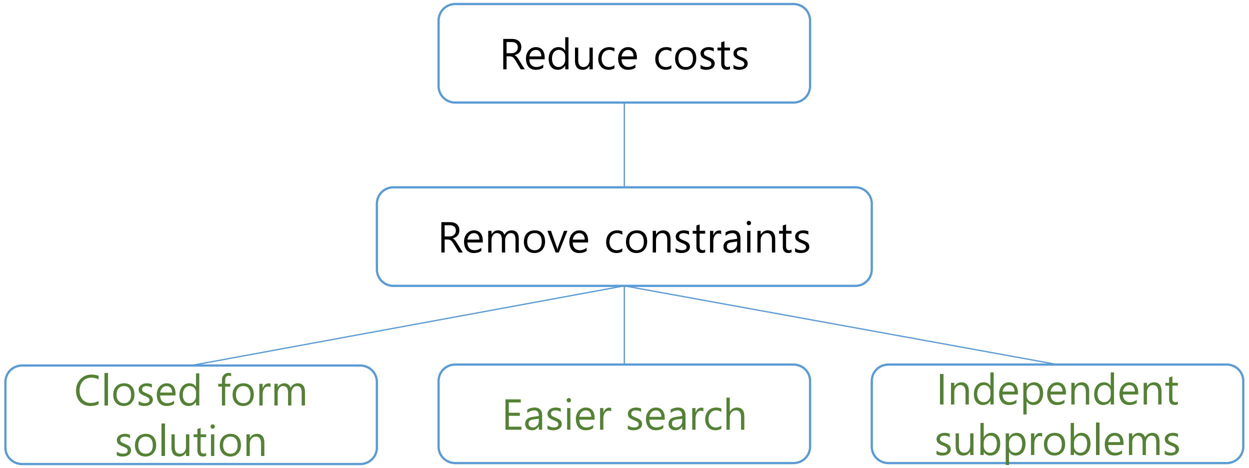
Relaxation

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

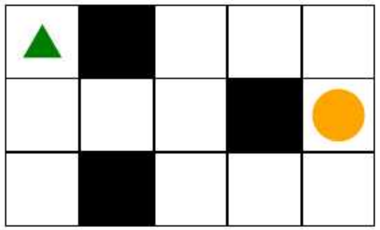


Combine heuristics using max

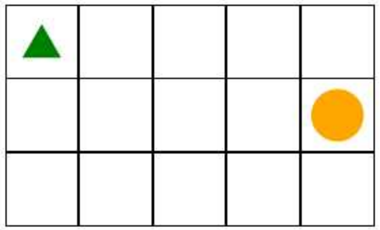
Closed form solution

Example: knock down walls

- Goal: move from triangle to circle



Hard



Easy

- Heuristic:

$$h(s) = \mathrm{ManhattanDistance}\big(s, (2,5)\big)$$
$$\mathrm{e.g.}, h((1,1)) = 5$$

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: 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~~

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

Relaxed state: location

Easier search

- Compute relaxed FutureCost'(location) for each location (1, …, n) using dynamic programming or UCS
- Modify UCS to compute all past costs in reversed relaxed problem (equivalent to future costs in relaxed problem!)

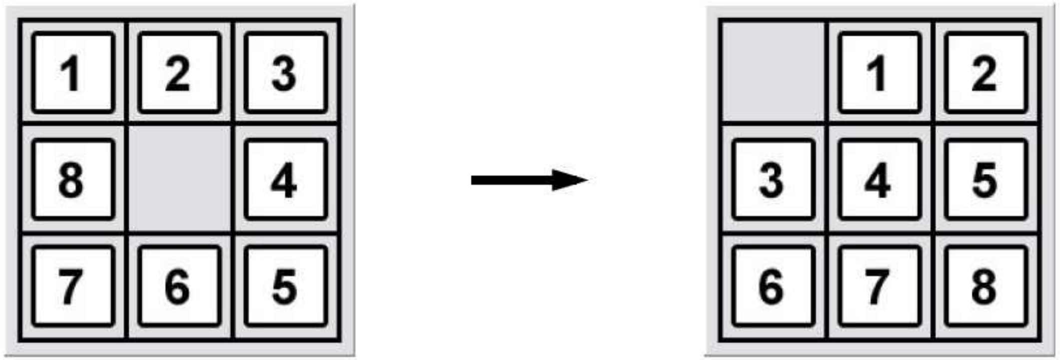
  Example: reversed relaxed problem
  - Start state: n
  - Walk action: from s to $s-1$ (cost: 1)
  - Tram action: from s to $s/2$ (cost: 2)
  - End state: 1

- Define heuristic for original problem:

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

Independent subproblems

[8 puzzle]



- Original problem: tiles cannot overlap (constraint)
- Relaxed problem: tiles can overlap (no constraint)
- Relaxed solution: 8 independent problems, each in closed form

Key idea: independence

- Relax original problem into independent subproblems

General framework

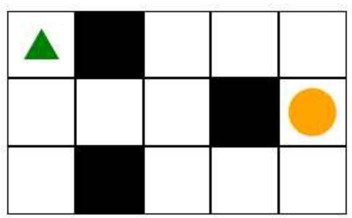
Removing constraints

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

Reducing edge costs

(from ∞ to some finite cost)

Example:



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

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

General framework

Definition: relaxed search problem

- A **relaxation** P' of a search problem P has costs that satisfy:

$$\text{Cost}'(s, a) \leq \text{Cost}(s, a)$$

Note: $\text{Cost}'(s, a)$ here is different from that in the definition of *consistent heuristics*: A heuristic h is **consistent** if $h(s_{\text{end}}) = 0$ and $\text{Cost}'(s, a) = \text{Cost}(s, a) + h\big(\text{Succ}(s, a)\big) - h(s) \geq 0$

Definition: relaxed heuristic

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

General framework

Theorem: consistency of relaxed heuristics

- Suppose $h(s) = \text{FutureCost}'(s)$ for some relaxed problem P'. Then $h(s)$ is a consistent heuristic.

Proof:

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

Tradeoff

Correctness:

- Edge costs are lower => consistent heuristic
- Example: removing constraints

Efficiency:

- $h(s) = \mathrm{FutureCost}'(s)$ must be easy to compute
- Examples: closed form, easier search, independent subproblems

Max of two heuristics

How do we combine two heuristics?

Proposition: max heuristics

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

Proof: exercise

Summary

- Structured Perceptron (reverse engineering): learn cost functions (search + learning)
- A*: add in heuristic estimate of future costs
- Relaxation (breaking the rules): framework for producing consistent heuristics
- Next time: when actions have unknown consequences…