Artificial Intelligence Foundation – JC3001

Lecture 29: Algorithms and Heuristics - II

Prof. Aladdin Ayesh (aladdin.ayesh@abdn.ac.uk)

Dr. Binod Bhattarai (binod.bhattarai@abdn.ac.uk)

Dr. Gideon Ogunniye, (g.ogunniye@abdn.ac.uk)

October 2025

Material adapted from:
Russell and Norvig (AIMA Book): Chapter 11 (11.2–11.3)
Jörg Hoffmann (University of Saarland)
Malte Helmert (University of Basel)
Dana Nau (University of Maryland)

UNIVERSITY OF ABERDEEN

Course Progression

- Part 1: Introduction
  1. Introduction to AI ✓
  2. Agents ✓
- Part 2: Problem-solving
  1. Search 1: Uninformed Search ✓
  2. Search 2: Heuristic Search ✓
  3. Search 3: Local Search ✓
  4. Search 4: Adversarial Search ✓
- Part 3: Reasoning and Uncertainty
  1. Reasoning 1: Constraint Satisfaction ✓
  2. Reasoning 2: Logic and Inference ✓
  3. Probabilistic Reasoning 1: BNs ✓
  4. Probabilistic Reasoning 2: HMMs ✓
- Part 4: Planning
  1. Planning 1: Intro and Formalism ✓
  2. **Planning 2: Algorithms and Heuristics**
  3. Planning 3: Hierarchical Planning
  4. Planning 4: Stochastic Planning
- Part 5: Learning
  1. Learning 1: Intro to ML
  2. Learning 2: Regression
  3. Learning 3: Neural Networks
  4. Learning 4: Reinforcement Learning
- Part 6: Conclusion
  1. Ethical Issues in AI
  2. Conclusions and Discussion

UNIVERSITY OF ABERDEEN

Objectives

- Planning Algorithms
  - Planning Graphs ✓
  - Planning based on Heuristic Search
  - Satisfiability Testing

Outline

- Heuristic Search Planning
- Heuristics for Classical Planning

From STRIPS Problem P to State Model $S(P)$

- A STRIPS problem P = $\langle$F,O,I,G$\rangle$ determines state model S(P) where
  - the states $s \in S$ are collections of atoms from F
  - the initial state s_0 is I
  - the goal states s are such that $G \subseteq s$
  - the actions a in $A(s)$ are ops in O s.t. $Pre(a) \subseteq s$
  - the next state is $s' = s - Del(a) + Add(a)$
  - action costs $c(a, s)$ are all 1
- How to solve $S(P)$?

- Explicitly **searches** graph associated with model $S(P)$ with **heuristic** $h(s)$ that estimates cost from s to goal
- **Key idea:** Heuristic h extracted **automatically** from problem P
- This is the mainstream approach in classical planning (and other forms of planning as well), enabling the solution of problems over **huge spaces**

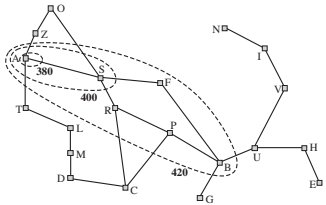
Heuristic Graph Search


```
function graphSearch(problem p, strategy s)
    closed ← {}                                   ▷ We now keep a list of explored states
    frontier.add(newNode(p.initial))
    loop
        if frontier is empty then
            return fail
        n ← s.removeChoice(frontier)
        closed.add(n.state)                       ▷ Where we keep states we already visisted
        if p.goalTest(n.state) then
            return getPath(n)
        for all a ∈ p.actions(n) do
            n′ ← a.result(n.state)
            if n′.state ∉ closed then
                frontier.add(n′)                  ▷ And only explore states we haven't visited
```

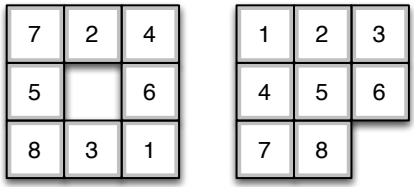
- In the case of tree search, A* is optimal **only** if $h(n)$ is an **admissible heuristic**
  - A heuristic is admissible, or "optimistic" if it **never overestimates** the cost to reach the goal
- Using such a heuristic means that $f(n)$ never overestimates the cost of reaching the goal
- The straight line distance is admissible. Why?

- Tree search with A* is optimal only if the heuristic used is admissible
  - Assume that there is suboptimal goal node G in the frontier, and let the cost of the optimal solution be C^*. Now, $f(G) = g(G) + h(G)$
  - Since $h(G) = 0$ (as h is admissible), $g(G) > C^*$
  - Now consider a different node on an optimal solution path. If h does not overestimate, then $f(n) = g(n) + h(n) < C^*$
  - So $f(n) \leq C^* < f(G)$ so G will not be expanded, and therefore A^* must return an optimal solution

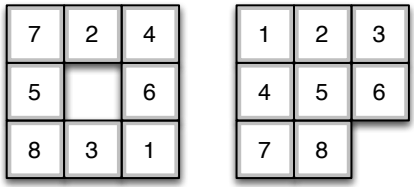
Contours



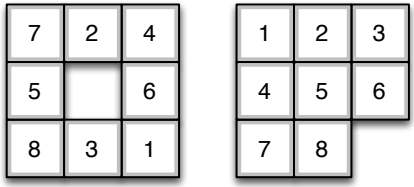
- Accurate heuristic functions cause the contour to stretch towards the optimal path more narrowly
- A* search is very good, but the number of nodes within the goal contour search space is still exponential in the length of the solution (unless the heuristic is very very good)
- There are other search techniques which use much less memory at the cost of increased computation time.
- For many problems, the optimality requirement must be dropped.



- Branching factor:
- Average depth:
- Total number of states:

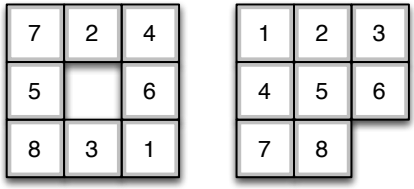


- Branching factor: 3
- Average depth: 22
- Total number of states: 3^{22}



- Heuristics:

- Are they admissible?



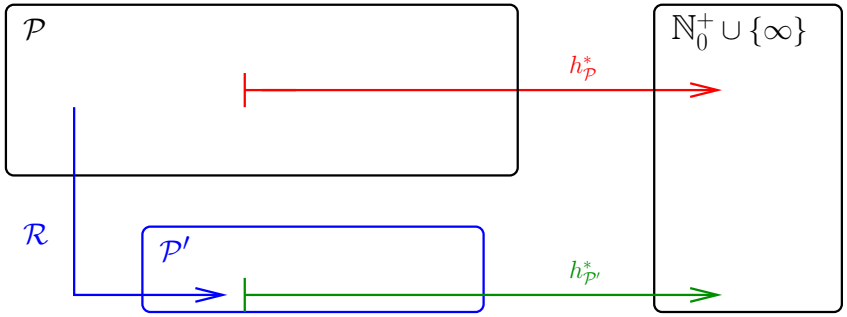
- Heuristics:
  - The number of misplaced tiles – **hamming distance** ($h_1 = 6$)
  - The sum of distances of the tiles from their goal positions – **manhattan distance** ($h_2 = 14$)
- Are they admissible?

- Ignore specific preconditions of actions
  (need some domain knowledge)
- Consider the sliding blocks from search

$$
\begin{aligned}
&Action(Slide(t, s_1, s_2),\\
&\quad \text{PRECOND:} On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)\\
&\quad \text{EFFECT:} On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2))
\end{aligned}
$$

- If we remove the $Blank(s_2)$, we are left with Manhattan distance

- h_2 is always greater than h_1
  - It therefore dominates h_1 (it is a tighter bound on the actual cost)
  - And is more efficient
- How can we invent good heuristics?
- Typically, by relaxing the problem.



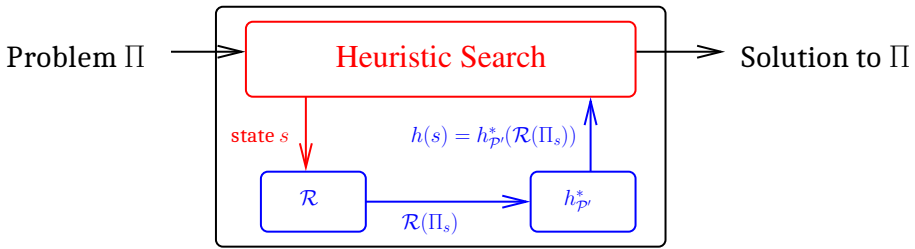
- You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
- You define a class $\mathcal{P}'$ of simpler problems, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to estimate $h^*_{\mathcal{P}}$.

How to Relax During Search: Overview

Search uses the real (un-relaxed) Π. The relaxation is applied **only within the call to $h(s)$!!!**

How to Relax During Search: Overview

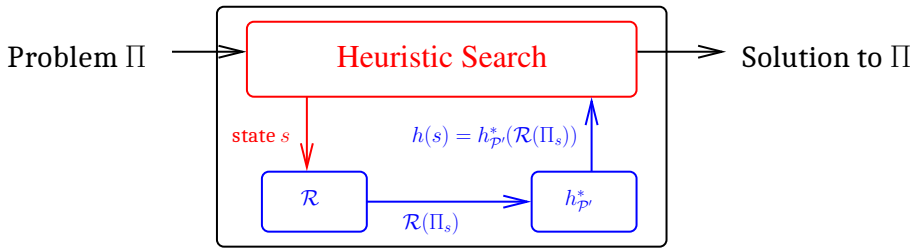
Search uses the real (un-relaxed) Π. The relaxation is applied **only within the call to $h(s)$!!!**



- Here, Π_s is Π with initial state replaced by s, i.e., $\Pi = (P, A, I, G)$ changed to (P, A, s, G): The task of finding a plan for search state s.

How to Relax During Search: Overview

Search uses the real (un-relaxed) Π. The relaxation is applied **only within the call to $h(s)$!!!**



- Here, Π_s is Π with initial state replaced by s, i.e., $\Pi = (P, A, I, G)$ changed to (P, A, s, G): The task of finding a plan for search state s.
- A common student mistake is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".

Outline

- Key development in planning in the 90's, is automatic extraction of **heuristic functions** to guide search for plans
- The general idea was known: heuristics often **explained** as **optimal** cost functions of **relaxed** (simplified) problems (Minsky 61; Pearl 83)
- Most common relaxation in planning, P^+, obtained by dropping delete-lists from ops in P. If $c^*(P)$ is optimal cost of P, then

$$h^+(P) \stackrel{\text{def}}{=} c^*(P^+)$$

- Heuristic h^+ **intractable** but easy to approximate; i.e.
  - computing **optimal** plan for P^+ is **intractable**, but
  - computing a non-optimal plan for P^+ **(relaxed plan)** easy
- State-of-the-art heuristics as in FF or LAMA still rely on P^+

- For all **atoms** p:

$$h(p;s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s, \text{else} \\ \min_{a \in O(p)} [cost(a) + h(pre(a);s)] \end{cases}$$

- For **sets** of atoms C, assume **independence**:

$$h(C;s) \stackrel{\text{def}}{=} \sum_{r \in C} h(r;s)$$

- Resulting **heuristic function** $h^{\text{add}}(s)$

$$h^{\text{add}}(s) \stackrel{\text{def}}{=} h(Goals;s)$$

- Heuristic not admissible, but informative and fast

- For all **atoms** p:

$$h(p;s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s, \text{ else} \\ \min_{a \in O(p)}[1 + h(pre(a);s)] \end{cases}$$

- For **sets** of atoms C, replace **sum** by **max**:

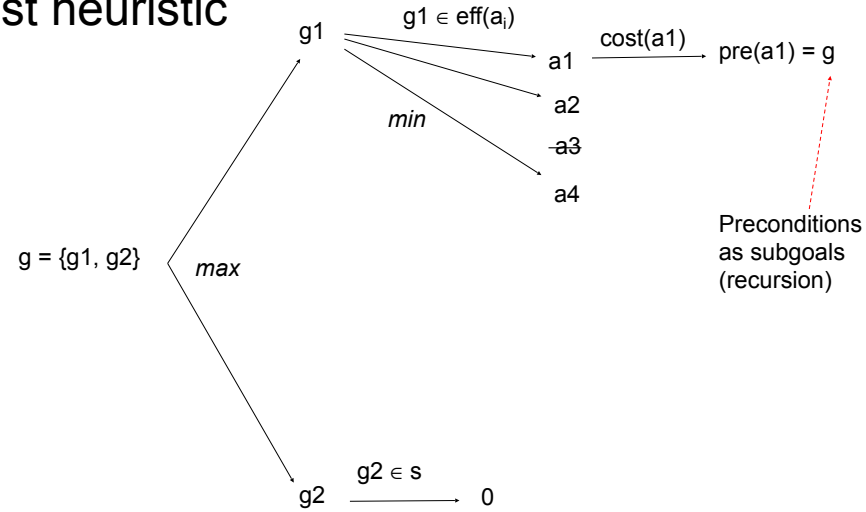
$$h(C;s) \stackrel{\text{def}}{=} \max_{r \in C} h(r;s)$$

- Resulting **heuristic function** $h^{\max}(s)$

$$h^{\max}(s) \stackrel{\text{def}}{=} h(Goals;s)$$

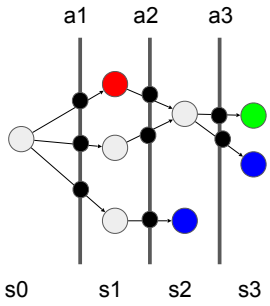
- Heuristic admissible, but not very informative

Max-cost heuristic



Max-cost heuristic for cost(a_i) = 1

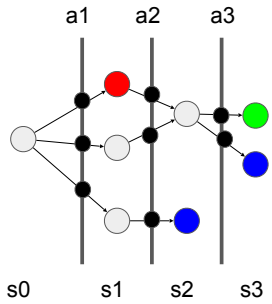
g = RGB



h^{max}(s0, RGB) = ?

Max-cost heuristic for cost(a_i) = 1

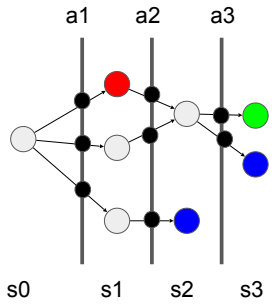
g = RGB



h^{max}(s0, RGB) = max(min(1), min(2,3), min(3))
= max(1, 2, 3) = 3

Max-cost heuristic for cost(a_i) = 1

g = RGB




```
define h^max(s_0, g)
  reachable ← copy(s_0)
  max ← 0
  while g - reachable ≠ ø
    last_state ← copy(reachable)
    for a ∈ A
      if pre(a) - last_state = ø
        reachable ← reachable U eff(a)
    if last_state = reachable
      return Infinity
    max ← max + 1
  return max
```

h^{max}(s0, RGB) = max(min(1), min(2,3), min(3))
= max(1, 2, 3) = 3

To continue in the next session.