Outline

- Motivation for Heuristics in Search algorithms
- A (non-exhaustive) overview on common heuristics for planning
- Some remarks on designing heuristics in practice
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Dijkstra’s algorithm:
  Uninformed search, explores
  space breadth-first

Imagine the difference in an n-dimensional grid!
Path finding in a uniform 2D grid

Dijkstra’s algorithm:
Uninformed search, explores space breadth-first

Imagine the difference in an n-dimensional grid!
Path finding in a uniform 2D grid

- Dijkstra’s algorithm:
  Uninformed search, explores space breadth-first

- Better: Use a heuristic to guide the search (e.g. best-first, $A^*$)

E.g. Euclidean distance to the goal as a lower bound of how much is left

Traversed subspace almost "one-dimensional", depending on search strategy and heuristic

Imagine the difference in an n-dimensional grid!
Path finding in a uniform 2D grid

Dijkstra’s algorithm:
Uninformed search, explores space breadth-first
Path finding in a uniform 2D grid

Dijkstra’s algorithm:
Uninformed search, explores space breadth-first

Better: Use a heuristic to guide the search (e.g. best-first, $A^*$)

E.g. Euclidean distance to the goal as a lower bound of how much is left

Traversed subspace almost "one-dimensional", depending on search strategy and heuristic

Imagine the difference in an $n$-dimensional grid!
Path finding in a uniform 2D grid

Dijkstra’s algorithm:
Uninformed search, explores space breadth-first

Better: Use a heuristic to guide the search (e.g. best-first, A∗)

E.g. Euclidean distance to the goal as a lower bound of how much is left

Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic

Imagine the difference in an n-dimensional grid!
Path finding in a uniform 2D grid

Dijkstra’s algorithm:
Uninformed search, explores space breadth-first
Path finding in a uniform 2D grid

Dijkstra’s algorithm:
Uninformed search, explores space breadth-first
Path finding in a uniform 2D grid

Dijkstra’s algorithm:
Uninformed search, explores space breadth-first

Imagine the difference in an n-dimensional grid!
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Better: Use a heuristic to guide the search (e.g. best-first, A*)
  - E.g. Euclidean distance to the goal as a lower bound of how much is left
  - Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Better: Use a heuristic to guide the search (e.g. best-first, A*)
  - E.g. Euclidean distance to the goal as a lower bound of how much is left
  - Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Better: Use a heuristic to guide the search (e.g. best-first, A*)
  - E.g. Euclidean distance to the goal as a lower bound of how much is left
  - Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Better: Use a **heuristic** to guide the search (e.g. best-first, A*)
  - E.g. **Euclidean distance to the goal** as a lower bound of how much is left
  - Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Better: Use a heuristic to guide the search (e.g. best-first, A*)
  - E.g. Euclidean distance to the goal as a lower bound of how much is left
  - Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Better: Use a **heuristic** to guide the search (e.g. best-first, A*)
  - E.g. **Euclidean distance to the goal** as a lower bound of how much is left
  - Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic
Heuristics: What we already know (1)

- Path finding in a uniform 2D grid
- Better: Use a heuristic to guide the search (e.g. best-first, A*)
  - E.g. Euclidean distance to the goal as a lower bound of how much is left
  - Traversed subspace almost “one-dimensional”, depending on search strategy and heuristic
- Imagine the difference in an n-dimensional grid!

Tomáš Balyo, Dominik Schreiber – Planning and Scheduling

November 15, 2018
Planning by State-Space Search is basically the same problem.
High-dimensional search space, action sequences are possible paths.

- Heuristics: What we already know (2)
Heuristics: What we already know (2)

- Planning by State-Space Search is basically the same problem
  - High-dimensional search space, action sequences are possible paths
- Can we apply the A* idea to domain-independent planning?
  - In A*, nodes are prioritized by their value $f(n) = c(n) + h(n)$
    - $c(n)$: Cost to get to this node
    - $h(n)$: Heuristic: Lower bound for the remaining cost to the goal
Planning by State-Space Search is basically the same problem. High-dimensional search space, action sequences are possible paths.

Can we apply the $A^*$ idea to domain-independent planning?

In $A^*$, nodes are prioritized by their value $f(n) = c(n) + h(n)$.

- $c(n)$: Cost to get to this node
- $h(n)$: Heuristic: Lower bound for the remaining cost to the goal

$c(n) := \text{sum of action cost so far}$

$= \text{sum of actions, if cost per action is constant}$

$h(n) := \text{euclidean distance to goal}$

What does this mean in state space?
A first heuristic for planning (1)

- Imagine state space as the corners of a $|\mathcal{P}|$-dimensional hypercube
- Every atom corresponds to one edge, its possible values true and false are the adjacent corners

How do we measure euclidean distances in $|\mathcal{P}|$-dimensions?

$$d(p, q) = \sqrt{\sum_{i=1}^{|\mathcal{P}|} (p_i - q_i)^2}$$

In our discrete sub-space: for two states $p$ and $q$, $(p_i - q_i)^2 = 1$ if they differ in the $i$-th atom, $(p_i - q_i)^2 = 0$, else.
A first heuristic for planning (1)

- Imagine state space as the corners of a $|\mathcal{P}|$-dimensional hypercube
  - Every atom corresponds to one edge, its possible values true and false are the adjacent corners
- How do we measure euclidean distances in $|\mathcal{P}|$ dimensions?

$$d(p, q) = \sqrt{\sum_{i=1}^{|\mathcal{P}|} (p_i - q_i)^2}$$

In our discrete sub-space: for two states $p$ and $q$, 
$$(p_i - q_i)^2 = 1 \text{ if they differ in the } i\text{-th atom, } (p_i - q_i)^2 = 0, \text{ else}$$

How do we measure the distance from a state $s$ to our goal $g$?

Only consider the sub-space involving the atoms in the goal
Leads to 
$$d(s, g) = \sqrt{\sum_{a \in g} \left[ a \in s \right]^2}$$
A first heuristic for planning (1)

- Imagine state space as the corners of a $|\mathcal{P}|$-dimensional hypercube.
  - Every atom corresponds to one edge, its possible values true and false are the adjacent corners.
- How do we measure euclidean distances in $|\mathcal{P}|$ dimensions?
  - $d(p, q) = \sqrt{\sum_{i=1}^{|\mathcal{P}|} (p_i - q_i)^2}$
  - In our discrete sub-space: for two states $p$ and $q$,
    - $(p_i - q_i)^2 = 1$ if they differ in the $i$-th atom,
    - $(p_i - q_i)^2 = 0$, else
Imagine state space as the corners of a $|\mathcal{P}|$-dimensional hypercube.

Every atom corresponds to one edge, its possible values true and false are the adjacent corners.

How do we measure euclidean distances in $|\mathcal{P}|$ dimensions?

$$d(p, q) = \sqrt{\sum_{i=1}^{|\mathcal{P}|} (p_i - q_i)^2}$$

In our discrete sub-space: for two states $p$ and $q$,

$$(p_i - q_i)^2 = 1 \text{ if they differ in the } i\text{-th atom},$$

$$(p_i - q_i)^2 = 0, \text{ else}$$

How do we measure the distance from a state $s$ to our goal $g$?
A first heuristic for planning (1)

- Imagine state space as the corners of a $|\mathcal{P}|$-dimensional hypercube
  - Every atom corresponds to one edge, its possible values true and false are the adjacent corners
- How do we measure euclidean distances in $|\mathcal{P}|$ dimensions?
  - $d(p, q) = \sqrt{\sum_{i=1}^{|\mathcal{P}|} (p_i - q_i)^2}$
  - In our discrete sub-space: for two states $p$ and $q$,
    - $(p_i - q_i)^2 = 1$ if they differ in the $i$-th atom,
    - $(p_i - q_i)^2 = 0$, else
- How do we measure the distance from a state $s$ to our goal $g$?
  - Only consider the sub-space involving the atoms in the goal
  - Leads to $d(s, g) := \sqrt{\sum_{a \in g} 1 - [a \in s]}$
A first heuristic for planning (2)

- Truck example
  - Goals: \( \text{at}(\text{package1}, \text{loc3}), \text{at}(\text{package2}, \text{loc3}), \text{at}(\text{package3}, \text{loc3}) \)
  - Initial state: None of above atoms are true
- Interpret initial state as point \( s_0 = (0, 0, 0) \) and goal as \( g = (1, 1, 1) \), one dimension per atom
- Initially, \( d(s, g) = d(s_0, g) \)
A first heuristic for planning (2)

- Truck example
  - Goals: \texttt{at(package1, loc3)}, \texttt{at(package2, loc3)}, \texttt{at(package3, loc3)}
  - Initial state: None of above atoms are true
- Interpret initial state as point $s_0 = (0, 0, 0)$ and goal as $g = (1, 1, 1)$, one dimension per atom
- Initially, $d(s, g) = d(s_0, g) = \sqrt{3}$
A first heuristic for planning (2)

- Truck example
  - Goals: \( \text{at}(\text{package}1, \text{loc}3) \), \( \text{at}(\text{package}2, \text{loc}3) \), \( \text{at}(\text{package}3, \text{loc}3) \)
  - Initial state: None of above atoms are true
- Interpret initial state as point \( s_0 = (0, 0, 0) \) and goal as \( g = (1, 1, 1) \), one dimension per atom
- Assume we bring \text{package}1 to \text{loc}3
A first heuristic for planning (2)

- Truck example
  - Goals: \( \text{at}(\text{package}1, \text{loc}3), \text{at}(\text{package}2, \text{loc}3), \text{at}(\text{package}3, \text{loc}3) \)
  - Initial state: None of above atoms are true
- Interpret initial state as point \( s_0 = (0, 0, 0) \) and goal as \( g = (1, 1, 1) \), one dimension per atom
- Assume we bring \text{package}1 to \text{loc}3
  \[ d(s, g) = \sqrt{2} \]
A first heuristic for planning (2)

- Truck example
  - Goals: at(package1, loc3), at(package2, loc3), at(package3, loc3)
  - Initial state: None of above atoms are true
  - Interpret initial state as point $s_0 = (0, 0, 0)$ and goal as $g = (1, 1, 1)$, one dimension per atom
  - Assume we also bring package2 to loc3
A first heuristic for planning (2)

- Truck example
  - Goals: at(package1, loc3), at(package2, loc3), at(package3, loc3)
  - Initial state: None of above atoms are true
- Interpret initial state as point $s_0 = (0, 0, 0)$ and goal as $g = (1, 1, 1)$, one dimension per atom
- Assume we also bring package2 to loc3
  $\Rightarrow d(s, g) = 1$
A first heuristic for planning (2)

- Truck example
  - Goals: at(package1, loc3),
    at(package2, loc3), at(package3, loc3)
  - Initial state: None of above atoms are true
- Interpret initial state as point \( s_0 = (0, 0, 0) \) and goal as \( g = (1, 1, 1) \), one dimension per atom
- When all packages are where they should be:
  \( d(s, g) = 0 \)
Admissible heuristics (1)

Admissibility.

A heuristic $h(s)$ regarding a goal $g$ is admissible iff $\forall s : h(s) \leq h^*(s)$, where $h^*(s)$ is the actual remaining distance to the goal $g$.

What does an admissible heuristic imply if using A*?
Admissible heuristics (1)

Admissibility.

A heuristic $h(s)$ regarding a goal $g$ is admissible iff $\forall s : h(s) \leq h^*(s)$, where $h^*(s)$ is the actual remaining distance to the goal $g$.

- What does an admissible heuristic imply if using $A^*$?
  - $\Rightarrow$ Optimal plan will be found

- Optimality in basic STRIPS planning: Shortest plan

- Optimality in planning with (non-uniform) action costs:
  - Plan with lowest total cost
    - Much more complex to find

- In the following, we mostly stay with uniform action cost
Admissible heuristics (2)

Admissibility.

A heuristic $h(s)$ regarding a goal $g$ is admissible iff $\forall s : h(s) \leq h^*(s)$, where $h^*(s)$ is the actual remaining distance to the goal $g$.

- Our heuristic: $h_{euc}(s) := d(s, g) = \sqrt{\sum_{a \in g} 1 - [a \in s]}
- Is this heuristic admissible (for uniform action costs)?
Admissible heuristics (2)

**Admissibility.**

A heuristic $h(s)$ regarding a goal $g$ is *admissible* iff $\forall s : h(s) \leq h^*(s)$, where $h^*(s)$ is the actual remaining distance to the goal $g$.

- Our heuristic: $h_{euc}(s) := d(s, g) = \sqrt{\sum_{a \in g} 1 - [a \in s]}$
- Is this heuristic *admissible* (for uniform action costs)? Only if no action satisfies more than one goal atom.
- Discussion: Is the heuristic *useful*?
Admissible heuristics (2)

Admissibility.

A heuristic $h(s)$ regarding a goal $g$ is **admissible** iff $\forall s : h(s) \leq h^*(s)$, where $h^*(s)$ is the actual remaining distance to the goal $g$.

- Our heuristic: $h_{euc}(s) := d(s, g) = \sqrt{\sum_{a \in g} 1 - [a \in s]}

- Is this heuristic **admissible** (for uniform action costs)?
  Only if no action satisfies more than one goal atom.

- Discussion: Is the heuristic **useful**?
  - Can be useful if problem has many goals, each of which only take a single action
  - Mostly useless when many actions are required to produce a goal
  - Worst Case: Large planning problems with a single goal; heuristic degenerates to $h_{euc}(s) = 1$ for almost all $s$
Good heuristics wanted

- Any ideas for better heuristics?
Any ideas for better heuristics?

- Finer grained metric than “amount of satisfied goals”
- Acknowledge non-goal atoms in the state as well
- Consider applicable actions and which changes they would bring
Good heuristics wanted

- Any ideas for better heuristics?
  - Finer grained metric than “amount of satisfied goals”
  - Acknowledge non-goal atoms in the state as well
  - Consider applicable actions and which changes they would bring

- Attributes of quality for a planning heuristic
  - Admissibility (although not always required)
  - Significance: should convey useful information of appropriate resolution
  - Accuracy: as close as possible to $h^*(s)$
  - Computability: Easier to compute than the original problem!
    Allowed complexities: Linear? Polynomial? Even NP-complete?
The central paradigm: Relaxation

- We cannot hope to find the true goal distance from all $s$ for our heuristic
  - Computing it already implies finding a path (= a plan)!
  - Even worse when dealing with action costs
The central paradigm: Relaxation

- We cannot hope to find the true goal distance from all $s$ for our heuristic
  - Computing it already implies finding a path (= a plan)!
  - Even worse when dealing with action costs
- Instead, fall back to a simplified problem (relaxation)
  - Easier to compute and analyze
  - Provides at least certain bounds for the original problem’s properties
- New objective: Find such a relaxation for planning problems
Delete-relaxation

Let \( \pi = (P, A, s_I, G) \) a planning problem. Then \( \pi_r := (P, A_r, s_I, G) \) is called the delete-relaxation of \( \pi \), whereas \( A_r = \{ a_r \mid a \in A \} \), with
\[
pre(a_r) = pre^+(a) \text{ and } eff(a_r) = eff^+(a).
\]
Delete-relaxation [GNT16]

Let \( \pi = (\mathcal{P}, \mathcal{A}, s_I, \mathcal{G}) \) a planning problem. Then \( \pi_r := (\mathcal{P}, \mathcal{A}_r, s_I, \mathcal{G}) \) is called the *delete-relaxation* of \( \pi \), whereas \( \mathcal{A}_r = \{ a_r \mid a \in \mathcal{A} \} \), with \( \text{pre}(a_r) = \text{pre}^+(a) \) and \( \text{eff}(a_r) = \text{eff}^+(a) \).

- Intuitively, \( \pi_r \) only considers **positive atoms** at each state
  - Negative preconditions and negative ("delete") effects of actions are neglected
  - Resulting problem corresponds to \( \text{PLANSAT}_+^+ \) (see lecture 2)
Delete-relaxation

Delete-relaxation [GNT16]

Let $\pi = (\mathcal{P}, \mathcal{A}, s_I, \mathcal{G})$ a planning problem. Then $\pi_r := (\mathcal{P}, \mathcal{A}_r, s_I, \mathcal{G})$ is called the delete-relaxation of $\pi$, whereas $\mathcal{A}_r = \{a_r \mid a \in \mathcal{A}\}$, with $\text{pre}(a_r) = \text{pre}^+(a)$ and $\text{eff}(a_r) = \text{eff}^+(a)$.

- Intuitively, $\pi_r$ only considers positive atoms at each state
- Negative preconditions and negative (“delete”) effects of actions are neglected
- Resulting problem corresponds to $\text{PLANSAT}_+^+$ (see lecture 2)
- Important property: All valid action sequences in $\pi$ from some state $s$ are also valid in $\pi_r$ from $s$
- Problem never gets harder in any way
Making use of delete-relaxed problems

What can we do with $\pi_r$?

- All effects are positive: Applying actions leads to increasing set of atoms until some fixpoint (or until all goals are reached)
Making use of delete-relaxed problems

What can we do with $\pi_r$?

- All effects are positive: Applying actions leads to increasing set of atoms until some fixpoint (or until all goals are reached)
- Construct relaxed planning graph
  - Planning graph: Alternation of states and applicable actions
  - Relaxation: Collapse possible states, execute all applicable actions at once
Making use of delete-relaxed problems

What can we do with $\pi_r$?

- All effects are positive: Applying actions leads to increasing set of atoms until some fixpoint (or until all goals are reached)
- Construct relaxed planning graph
  - Planning graph: Alternation of states and applicable actions
  - Relaxation: Collapse possible states, execute all applicable actions at once
Making use of delete-relaxed problems

What can we do with $\pi_r$?

- All effects are positive: Applying actions leads to increasing set of atoms until some fixpoint (or until all goals are reached)
- Construct relaxed planning graph
  - Planning graph: Alternation of states and applicable actions
  - Relaxation: Collapse possible states, execute all applicable actions at once
Making use of delete-relaxed problems

What can we do with $\pi_r$?

- All effects are positive: Applying actions leads to increasing set of atoms until some fixpoint (or until all goals are reached)
- Construct relaxed planning graph
  - Planning graph: Alternation of states and applicable actions
  - Relaxation: Collapse possible states, execute all applicable actions at once
Delete-relaxation: A first heuristic

Let’s assemble a heuristic:

- Given a state $s$, begin a relaxed planning graph starting at $s$
- Build graph layer for layer until all (positive) goals are reached
- Let $h_d^+(s) :=$ the required depth, i.e. amount of needed iterations
  - What if goals are not reached in the graph?

Discussion: Admissible? Useful? Easy to compute?

Admissible: only for action cost $\geq 1$ (see exercises)

Useful if relaxed problem has long critical path leading to the goals

Less useful if relaxed planning graph is very shallow

E.g. if almost all action preconditions are negative

Computational complexity: Easy (see complexity of PLANSAT)
Delete-relaxation: A first heuristic

Let’s assemble a heuristic:

- Given a state \( s \), begin a relaxed planning graph starting at \( s \)
- Build graph layer for layer until all (positive) goals are reached
- Let \( h_d^+ (s) \) := the required depth, i.e. amount of needed iterations
  - What if goals are not reached in the graph? \( \Rightarrow h_d^+ (s) = \infty \)
Delete-relaxation: A first heuristic

Let’s assemble a heuristic:

- Given a state \(s\), begin a relaxed planning graph starting at \(s\)
- Build graph layer for layer until all (positive) goals are reached
- Let \(h_d^+(s) := \text{the required depth}\), i.e. amount of needed iterations
  - What if goals are not reached in the graph? \(\Rightarrow h_d^+(s) = \infty\)

Discussion: Admissible? Useful? Easy to compute?
Delete-relaxation: A first heuristic

Let’s assemble a heuristic:

- Given a state \( s \), begin a relaxed planning graph starting at \( s \)
- Build graph layer for layer until all (positive) goals are reached
- Let \( h_d^+(s) := \) the required depth, i.e. amount of needed iterations
  - What if goals are not reached in the graph? \( \Rightarrow h_d^+(s) = \infty \)

Discussion: Admissible? Useful? Easy to compute?

- Admissible: only for action cost \( \geq 1 \) (see exercises)
Delete-relaxation: A first heuristic

Let’s assemble a heuristic:

- Given a state \( s \), begin a relaxed planning graph starting at \( s \)
- Build graph layer for layer until all (positive) goals are reached
- Let \( h_d^+(s) := \text{the required depth} \), i.e. amount of needed iterations
  - What if goals are not reached in the graph? \( \Rightarrow h_d^+(s) = \infty \)

Discussion: Admissible? Useful? Easy to compute?

- Admissible: only for action cost \( \geq 1 \) (see exercises)
- Useful if relaxed problem has long critical path leading to the goals
- Less useful if relaxed planning graph is very shallow
  - E.g. if almost all action preconditions are negative
Delete-relaxation: A first heuristic

Let’s assemble a heuristic:

- Given a state $s$, begin a relaxed planning graph starting at $s$.
- Build graph layer for layer until all (positive) goals are reached.
- Let $h_d^+(s) :=$ the required depth, i.e. amount of needed iterations.
  - What if goals are not reached in the graph? $\Rightarrow h_d^+(s) = \infty$

Discussion: Admissible? Useful? Easy to compute?

- Admissible: only for action cost $\geq 1$ (see exercises).
- Useful if relaxed problem has long critical path leading to the goals.
- Less useful if relaxed planning graph is very shallow.
  - E.g. if almost all action preconditions are negative.
- Computational complexity: Easy (see complexity of PLANSAT$^+$).
Delete-relaxation: A first heuristic

Let’s assemble a heuristic:

- Given a state \( s \), begin a relaxed planning graph starting at \( s \)
- Build graph layer for layer until all (positive) goals are reached
- Let \( h_d^+(s) := \text{the required depth} \), i.e. amount of needed iterations
  - What if goals are not reached in the graph? \( \Rightarrow h_d^+(s) = \infty \)

Discussion: Admissible? Useful? Easy to compute?

- Admissible: only for action cost \( \geq 1 \) (see exercises)
- Useful if relaxed problem has long critical path leading to the goals
- Less useful if relaxed planning graph is very shallow
  - E.g. if almost all action preconditions are negative
- Computational complexity: Easy (see complexity of PLANSAT\(^+\))

Intermission: \( h_d^+(s) \) in Aquaplanning
The Fast-Forward Heuristic (1)

- $h_d^+$ is inaccurate / has low resolution:
  few information on actual amount of required actions
The Fast-Forward Heuristic (1)

- $h^+_d$ is inaccurate / has low resolution: few information on actual amount of required actions
- More accurate: Fast-Forward heuristic $h^{FF}(s)$ [HN01]
  - Build relaxed planning graph $G$ from $s$ until relaxed goal $g^+$ is satisfied
  - (*) Extract an actual (relaxed) solution plan $p$
  - Return cost of $p$
The Fast-Forward Heuristic (1)

- $h_d^+$ is inaccurate / has low resolution:
  few information on actual amount of required actions
- More accurate: Fast-Forward heuristic $h^{FF}(s)$ [HN01]
  - Build relaxed planning graph $G$ from $s$
    until relaxed goal $g^+$ is satisfied
  - (*) Extract an actual (relaxed) solution plan $p$
  - Return cost of $p$

- Important idea for (*): If one can execute multiple actions $A$
  at the same state, the ordering of actions does not matter
  - Reasoning:
    1. All preconditions and effects are positive
    2. All actions are already applicable
    $\Rightarrow$ None of the actions will disable another one
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. \( \hat{s} := 1^{st} \) state in \( G \) which satisfies \( g^+; \) \( \hat{g} := g^+; \) \( p := \langle \rangle; \)
2. While \( \hat{s} \neq s: \)
3. \( \hat{s} := \) predecessor of \( \hat{s} \) in \( G; \)
4. Choose set of actions \( A \) applicable in \( \hat{s} \) such that \( \gamma^+(\hat{s}, A) \) satisfies \( \hat{g}; \)
5. \( p := A \circ p; \)
6. \( \hat{g} := (\hat{g} \setminus eff^+(A)) \cup pre^+(A); \)
7. Output \( p. \)

(Apply positive effects only)
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. \( \hat{s} := \) 1\textsuperscript{st} state in \( G \) which satisfies \( g^+; \)  \( \hat{g} := g^+; \)  \( p := \langle \rangle; \)
2. While \( \hat{s} \neq s; \)
3. \( \hat{s} := \) predecessor of \( \hat{s} \) in \( G; \)
4. Choose set of actions \( A \) applicable in \( \hat{s} \) such that \( \gamma^+(\hat{s}, A) \) satisfies \( \hat{g}; \)
5. \( p := A \circ p; \) \hfill (Apply positive effects only)
6. \( \hat{g} := (\hat{g} \setminus eff^+(A)) \cup pre^+(A); \)
7. Output \( p. \)
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. $\hat{s} := 1^{\text{st}} \text{ state in } G \text{ which satisfies } g^+; \quad \hat{g} := g^+; \quad p := \langle \rangle$;
2. While $\hat{s} \neq s$:
   
   3. $\hat{s} := \text{predecessor of } \hat{s} \text{ in } G$;
4. Choose set of actions $A$ applicable in $\hat{s}$ such that $\gamma^+(\hat{s}, A)$ satisfies $\hat{g}$;
5. $p := A \circ p$;
6. $\hat{g} := (\hat{g} \setminus \text{eff}^+(A)) \cup \text{pre}^+(A)$;

(Apply positive effects only)
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. \( \hat{s} := \text{1st state in } G \text{ which satisfies } g^+; \) \( \hat{g} := g^+; \) \( p := \langle \rangle; \)
2. While \( \hat{s} \neq s: \)
3. \( \hat{s} := \text{predecessor of } \hat{s} \text{ in } G; \)
4. Choose set of actions \( A \) applicable in \( \hat{s} \) such that \( \gamma^+(\hat{s}, A) \) satisfies \( \hat{g}; \)
5. \( p := A \circ p; \) \( \text{(Apply positive effects only)} \)
6. \( \hat{g} := (\hat{g} \setminus \text{eff}^+(A)) \cup \text{pre}^+(A); \)
7. Output \( p. \)
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. \( \hat{s} := 1^{st} \text{ state in } G \text{ which satisfies } g^+; \quad \hat{g} := g^+; \quad p := \langle \rangle; \)
2. While \( \hat{s} \neq s: \)
3. \( \hat{s} := \text{predecessor of } \hat{s} \text{ in } G; \)
4. Choose set of actions \( A \) applicable in \( \hat{s} \) such that \( \gamma^+(\hat{s}, A) \) satisfies \( \hat{g} \);
5. \( p := A \circ p; \)
6. \( \hat{g} := (\hat{g} \setminus \text{eff}^+(A)) \cup \text{pre}^+(A); \)
7. Output \( p. \)
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. $\hat{s} := 1^{\text{st}} \text{ state in } G \text{ which satisfies } g^+$; $\hat{g} := g^+$; $p := \langle \rangle$;
2. While $\hat{s} \neq s$:
3. $\hat{s} := \text{predecessor of } \hat{s} \text{ in } G$;
4. Choose set of actions $A$ applicable in $\hat{s}$ such that $\gamma^+(\hat{s}, A)$ satisfies $\hat{g}$;
5. $p := A \circ p$;
6. $\hat{g} := (\hat{g} \setminus \text{eff}^+(A)) \cup \text{pre}^+(A)$;

(Apply positive effects only)
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. \( \hat{s} := 1^{\text{st}} \) state in \( G \) which satisfies \( g^+ \);
2. \( \hat{g} := g^+ \);
3. \( p := \langle \rangle \);
4. While \( \hat{s} \neq s \):
   5. \( \hat{s} := \) predecessor of \( \hat{s} \) in \( G \);
   6. Choose set of actions \( A \) applicable in \( \hat{s} \) such that \( \gamma^+(\hat{s}, A) \) satisfies \( \hat{g} \);
   7. \( p := A \circ p \);
6. \( \hat{g} := (\hat{g} \setminus \text{eff}^+(A)) \cup \text{pre}^+(A) \);
7. Output \( p \).
The Fast-Forward Heuristic (2)

How to compute the relaxed plan?

1. \( \hat{s} := 1^{st} \) state in \( G \) which satisfies \( g^+ \); \( \hat{g} := g^+ \); \( p := \langle \rangle \);
2. While \( \hat{s} \neq s \):
3. \( \hat{s} := \) predecessor of \( \hat{s} \) in \( G \);
4. Choose set of actions \( A \) applicable in \( \hat{s} \) such that \( \gamma^+(\hat{s}, A) \) satisfies \( \hat{g} \);
5. \( p := A \circ p \);
6. \( \hat{g} := (\hat{g} \setminus \text{eff}^+(A)) \cup \text{pre}^+(A) \);
7. Output \( p \).

\[ \Rightarrow p = \langle \text{drive}(B), \text{pick-up}(p1), \text{pick-up}(p2), \text{drive}(C), \text{drop}(p1), \text{drop}(p2) \rangle \]
Properties of $h^{FF}$

When is $h^{FF}$ admissible?

- $h^{FF}$ admissible $\iff \forall s : h^{FF}(s) \leq h^*(s)$
  $\iff$ cost of relaxed extracted $p \leq$ cost of exact plan $p^*$
  $\iff$ $p$ never “overestimates” $p^*$

Uniform action costs: always pick minimum set of actions $A$
Equivalent to set cover problem (NP-complete)

Non-uniform action costs: always pick optimal set of actions $A$
Equivalent to weighted set cover problem (NP-complete)

Consequence:
$h^{FF}(s)$ can’t be both admissible and easy to compute

Essential cause: Finding an optimal plan is difficult even in delete-relaxed problems (Does this look familiar?)
Properties of $h^{FF}$

When is $h^{FF}$ admissible?

- $h^{FF}$ admissible $\iff \forall s : h^{FF}(s) \leq h^*(s)$
- $\iff$ cost of relaxed extracted $p \leq$ cost of exact plan $p^*$
- $\iff p$ never “overestimates” $p^*$

- Uniform action costs: always pick minimum set of actions $A$
  - Equivalent to set cover problem (NP-complete)

- Non-uniform action costs: always pick optimal set of actions $A$
  - Equivalent to weighted set cover problem (NP-complete)
Properties of $h^{FF}$

When is $h^{FF}$ admissible?

- $h^{FF}$ admissible $\iff \forall s : h^{FF}(s) \leq h^*(s)$
  $\iff$ cost of relaxed extracted $p \leq$ cost of exact plan $p^*$
  $\iff$ $p$ never “overestimates” $p^*$

- Uniform action costs: always pick minimum set of actions $A$
  - Equivalent to set cover problem (NP-complete)

- Non-uniform action costs: always pick optimal set of actions $A$
  - Equivalent to weighted set cover problem (NP-complete)

Consequence: $h^{FF}(s)$ can’t be both admissible and easy to compute

- Essential cause: Finding an optimal plan is difficult even in delete-relaxed problems (Does this look familiar?)
FF: Concluding remarks

Original implementation of $h^{FF}$ in the Fast-Forward planner [HN01]:

- Works with general action costs
- Choice of actions is approximated to be near-optimal
Original implementation of $h^{FF}$ in the Fast-Forward planner [HN01]:

- Works with general action costs
- Choice of actions is **approximated** to be near-optimal
- Heuristic search complemented by many additional techniques contributing to the success of FF
  - **Enforced hill climbing** (see last lecture)
  - Pruning of unpromising nodes:
    - Only apply actions **considered helpful** regarding the relaxed solution
  - Much more
Delete-relaxed Landmark heuristics

Crude idea of Landmarks:

- Try to find common points which all valid plans share
- How many such points are still missing? ⇒ Use for heuristic
Delete-relaxed Landmark heuristics

Crude idea of Landmarks:
- Try to find common points which all valid plans share
- How many such points are still missing? ⇒ Use for heuristic

Disjunctive Action Landmarks [HR15]

A disjunctive action landmark is a set of actions \( A \) such that each delete-relaxed plan contains some \( a \in A \).
Delete-relaxed Landmark heuristics

Crude idea of Landmarks:

- Try to find common points which all valid plans share
- How many such points are still missing? ⇒ Use for heuristic

Disjunctive Action Landmarks [HR15]

A *disjunctive action landmark* is a set of actions \( A \) such that each delete-relaxed plan contains some \( a \in A \).

Landmarks in Trucking example:

- \( L_1 = \{ \text{pickup}(1), \text{pickup}(2) \} \)
- \( L_2 = \{ \text{move}(B) \}, \ L_3 = \{ \text{move}(C) \} \)
Delete-relaxed Landmark heuristics

Crude idea of Landmarks:
- Try to find common points which all valid plans share
- How many such points are still missing? ⇒ Use for heuristic

Disjunctive Action Landmarks [HR15]

A *disjunctive action landmark* is a set of actions $A$ such that each delete-relaxed plan contains some $a \in A$.

Landmarks in Trucking example:
- $L_1 = \{\text{pickup}(1), \text{pickup}(2)\}$
- $L_2 = \{\text{move}(B)\}, L_3 = \{\text{move}(C)\}$
- Assume two alternative routes $B_1$ and $B_2$ from $B$ to $C$: $L_4 = \{\text{move}(B_1), \text{move}(B_2)\}$
Computing Landmarks

- Construct a **Justification Graph** [HR15]
  - Vertices: Single atoms
  - Directed edges for each action \( a \): **some precondition** of \( a \) to **each of its effects** (needs some precondition choice strategy)

Modified Trucking example (1 package, road \( B \rightarrow \{B_1, B_2\} \rightarrow C \)):
Computing Landmarks

- Construct a **Justification Graph** [HR15]
  - Vertices: Single atoms
  - Directed edges for each action \( a \): some precondition of \( a \) to each of its effects (needs some precondition choice strategy)
- Modified Trucking example (1 package, road \( B \rightarrow \{B_1, B_2\} \rightarrow C\)):
Computing Landmarks

- Construct a **Justification Graph** [HR15]
  - Vertices: Single atoms
  - Directed edges for each action \( a \): *some precondition of \( a \) to each of its effects* (needs some precondition choice strategy)
- Modified Trucking example (1 package, road \( B \rightarrow \{ B_1, B_2 \} \rightarrow C \)):

  - truck-at(B)
  - truck-at(B_1)
  - truck-at(B_2)
  - truck-at(C)
  - at(p1,C)
  - move(B,B_1)
  - move(B,B_2)
  - move(B_1,C)
  - move(B_2,C)
  - drop(p1)

- Each **graph cut** (separating start from goal) forms a Landmark!
  - At least one of cut actions required to get to goal
  - Delete-relaxation, choice of preconditions **never increase cut size**
Computing Landmarks

- Construct a Justification Graph [HR15]
  - Vertices: Single atoms
  - Directed edges for each action $a$: some precondition of $a$
    to each of its effects (needs some precondition choice strategy)
- Modified Trucking example (1 package, road $B \rightarrow \{B_1, B_2\} \rightarrow C$):

  - Each graph cut (separating start from goal) forms a Landmark!
    - At least one of cut actions required to get to goal
    - Delete-relaxation, choice of preconditions never increase cut size
- Open question: Which preconditions to choose?
Computing Landmarks

- Construct a **Justification Graph** [HR15]
  - Vertices: Single atoms
  - Directed edges for each action $a$: some precondition of $a$
to each of its effects (needs some precondition choice strategy)

- Modified Trucking example (1 package, road $B \rightarrow \{B_1, B_2\} \rightarrow C$):

  - Each **graph cut** (separating start from goal) forms a Landmark!
    - At least one of cut actions required to get to goal
    - Delete-relaxation, choice of preconditions never increase cut size

- Open question: Which preconditions to choose?
Using Landmarks

- Suppose multiple landmarks $L = \{L_1, L_2, \ldots, L_k\}$ have been found from state $s$ to $g$
- How to exploit this knowledge for heuristic value $h_L(s)$?
Using Landmarks

- Suppose multiple landmarks $L = \{L_1, L_2, \ldots, L_k\}$ have been found from state $s$ to $g$.

- How to exploit this knowledge for heuristic value $h_L(s)$?
  - $h_L(s) := |L|$, more general: $h_L(s) := \sum_{L_i \in L} \left( \sum_{a \in L_i} \text{cost}(a) \right)$.
Using Landmarks

- Suppose multiple landmarks $L = \{L_1, L_2, \ldots, L_k\}$ have been found from state $s$ to $g$
- How to exploit this knowledge for heuristic value $h_L(s)$?
  - $h_L(s) := |L|$, more general: $h_L(s) := \sum_{L_i \in L} (\sum_{a \in L_i} \text{cost}(a))$
  - $h_L(s) := \max_{L_i \in L} (\sum_{a \in L_i} \text{cost}(a))$

Use Linear Programming approximating max. admissible cost for $L$

Find actual minimum set of cut actions with Minimum Hitting Set

$h_L(s)$ can be admissible; again, trade-off between accuracy and computational effort
Using Landmarks

- Suppose multiple landmarks $L = \{L_1, L_2, \ldots, L_k\}$ have been found from state $s$ to $g$

- How to exploit this knowledge for heuristic value $h_L(s)$?
  - $h_L(s) := |L|$, more general: $h_L(s) := \sum_{L_i \in L} (\sum_{a \in L_i} \text{cost}(a))$
  - $h_L(s) := \max_{L_i \in L} (\sum_{a \in L_i} \text{cost}(a))$
  - Use Linear Programming approximating max. admissible cost for $L$
  - Find actual minimum set of cut actions with Minimum Hitting Set
Using Landmarks

- Suppose multiple landmarks \( L = \{L_1, L_2, \ldots, L_k\} \) have been found from state \( s \) to \( g \).
- How to exploit this knowledge for heuristic value \( h_L(s) \)?
  - \( h_L(s) := |L| \), more general: \( h_L(s) := \sum_{L_i \in L} (\sum_{a \in L_i} \text{cost}(a)) \)
  - \( h_L(s) := \max_{L_i \in L} (\sum_{a \in L_i} \text{cost}(a)) \)
- Use Linear Programming approximating max. admissible cost for \( L \).
- Find actual minimum set of cut actions with Minimum Hitting Set.

- \( h_L(s) \) can be admissible;
- again, trade-off between accuracy and computational effort.
Relaxation by abstraction (1)

- Another relaxation: Abstract state space of some atom pattern $X$
- Example: Consider subspace of pattern $X := \{at(p_1, \cdot), at(p_2, \cdot)\}$
Relaxation by abstraction (1)

- Another relaxation: Abstract state space of some atom pattern $X$
- Example: Consider subspace of pattern $X := \{ at(p1, \cdot), at(p2, \cdot) \}$
Relaxation by abstraction (1)

- Another relaxation: Abstract state space of some atom pattern $X$
- Example: Consider subspace of pattern $X := \{at(p1, \cdot), at(p2, \cdot)\}$
Relaxation by abstraction (1)

- Another relaxation: **Abstract state space of some atom pattern** \( X \)
- Example: Consider subspace of pattern \( X := \{ at(p_1, \cdot), at(p_2, \cdot) \} \)

- Plan search in abstracted state space can be much easier
  \( \Rightarrow \) Exploit this for a heuristic
Relaxation by abstraction (2)

- Construction of abstraction-relaxed problem: easy
  - Remove (or neglect) all occurrences of un-picked atoms in actions, initial state and goal
Relaxation by abstraction (2)

- Construction of abstraction-relaxed problem: easy
  - Remove (or neglect) all occurrences of un-picked atoms in actions, initial state and goal
- Search through abstract state space for a plan $p_{abs}$:
  - Usual search techniques (hopefully easier than before!)
  - Build exhaustive pattern database:
    compute and store a plan for each abstract state
Relaxation by abstraction (2)

- Construction of abstraction-relaxed problem: easy
  - Remove (or neglect) all occurrences of un-picked atoms in actions, initial state and goal
- Search through abstract state space for a plan $p_{abs}$:
  - Usual search techniques (hopefully easier than before!)
  - Build exhaustive pattern database:
    compute and store a plan for each abstract state
- Heuristic value: mainly $h_{abs}(s) := \text{cost}(p_{abs})$
Relaxation by abstraction (2)

- Construction of abstraction-relaxed problem: easy
  - Remove (or neglect) all occurrences of un-picked atoms in actions, initial state and goal
- Search through abstract state space for a plan $p_{abs}$:
  - Usual search techniques (hopefully easier than before!)
  - Build exhaustive pattern database:
    - compute and store a plan for each abstract state
- Heuristic value: mainly $h_{abs}(s) := cost(p_{abs})$
- Which pattern $X$ to pick?
  - $X = g \Rightarrow h_{abs}(s)$ measures distance to goal, like $h_{euc}(s)$
Relaxation by abstraction (2)

- Construction of abstraction-relaxed problem: easy
  - Remove (or neglect) all occurrences of un-picked atoms in actions, initial state and goal
- Search through abstract state space for a plan $p_{abs}$:
  - Usual search techniques (hopefully easier than before!)
  - Build exhaustive pattern database:
    compute and store a plan for each abstract state
- Heuristic value: mainly $h_{abs}(s) := cost(p_{abs})$
- Which pattern $X$ to pick?
  - $X = g \Rightarrow h_{abs}(s)$ measures distance to goal, like $h_{euc}(s)$
  - Pick $X$ in a way that interconnected components in state space collapse
    - Example: Agent can freely move on a graph of waypoints
      $\Rightarrow$ Abstract away position of the agent
Abstraction heuristics in practice

Realizations of abstraction heuristics:

- Merge-and-shrink abstraction [HHHN14]
- Structural-pattern abstraction [KD09]

General properties:

- Hard part is to identify a good pattern $X$
- *Can be admissible*, if relaxed plan optimal
- State space explodes (again) if using too big $X$
Critical Path Heuristics

- Suppose we can apply multiple actions in parallel
  - Build causal dependency graph as DAG from initial state to goal
  - (At least) one path in DAG is the longest $\Rightarrow$ Critical path

Start

pick-up(p1)

move(B)

move(C)

pick-up(p2)

drop(C,p1)

drop(C,p2)

Goal
Critical Path Heuristics

- Suppose we can apply multiple actions in parallel
  - Build *causal dependency graph* as DAG from initial state to goal
  - (At least) one path in DAG is the longest \(\Rightarrow\) Critical path

```
pick-up(p1)  move(B)  move(C)  pick-up(p2)  drop(C,p2)  drop(C,p1)  Goal
Start
```
Critical Path Heuristics

- Suppose we can apply multiple actions in parallel
  - Build causal dependency graph as DAG from initial state to goal
  - (At least) one path in DAG is the longest $\Rightarrow$ Critical path

![Diagram of causal dependency graph]

- Idea for relaxation: Allow applying $\leq m$ actions in parallel
The \( h^m \) Heuristics Family

- Recursive definition of functions \( \{ h^m \mid m = 1, 2, \ldots \} \) [GH00], simplified to uniform action costs:

\[
h^m(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s. \\
\min_{a \in \text{Rel}(g)} \left( 1 + h^m(s, \gamma^{-1}(g, a)) \right), & \text{if } |g| \leq m. \\
\max_{g' \subseteq g, |g'| \leq m} h^m(s, g'), & \text{otherwise.}
\end{cases}
\]

- \text{Rel}(g): Relevant actions which have atoms in \( g \) as effects
- 3\textsuperscript{rd} case picks most costly \( m \)-subset of goal atoms
The $h^m$ Heuristics Family

- Recursive definition of functions $\{ h^m \mid m = 1, 2, \ldots \}$ [GH00], simplified to uniform action costs:

$$h^m(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s. \\
\min_{a \in \text{Rel}(g)} \left( 1 + h^m(s, \gamma^{-1}(g, a)) \right), & \text{if } |g| \leq m. \\
\max_{g' \subseteq g, |g'| \leq m} h^m(s, g'), & \text{otherwise}. 
\end{cases}$$

- $\text{Rel}(g)$: Relevant actions which have atoms in $g$ as effects
- 3$^\text{rd}$ case picks most costly $m$-subset of goal atoms
- Pick $m$, then $h_{\text{crit}}(s) := h^m(s, g)$

Properties:
- Admissible for all $m$ (crit. path never longer than optimal seq. plan!)
- Complexity polynomial in problem size for fixed $m$
- Complexity exponential for general $m$ (3$^\text{rd}$ case explodes)

Tomáš Balyo, Dominik Schreiber – Planning and Scheduling
November 15, 2018
The $h^m$ Heuristics Family

- Recursive definition of functions $\{h^m \mid m = 1, 2, \ldots\}$ [GH00], simplified to uniform action costs:

$$h^m(s, g) = \begin{cases} 
0, & \text{if } g \subseteq s. \\
\min_{a \in \text{Rel}(g)} \left(1 + h^m(s, \gamma^{-1}(g, a))\right), & \text{if } |g| \leq m. \\
\max_{g' \subseteq g, |g'| \leq m} h^m(s, g'), & \text{otherwise.}
\end{cases}$$

- $\text{Rel}(g)$: Relevant actions which have atoms in $g$ as effects
- 3rd case picks most costly $m$-subset of goal atoms
- Pick $m$, then $h_{\text{crit}}(s) := h^m(s, g)$
- Properties:
  - Admissible for all $m$ (crit. path never longer than optimal seq. plan!)
  - Complexity polynomial in problem size for fixed $m$
  - Complexity exponential for general $m$ (3rd case explodes)
Heuristics: Recap

Coarse overview on common heuristics for planning

- Central paradigm: Relaxation
- Heuristics based on Delete-relaxation:
  - Easy heuristics measuring goal distance
  - Fast-Forward heuristic $h^{FF}$
  - Family of Landmark heuristics
- Relaxation by abstraction: Heuristics on pattern databases
- Relaxation by parallelism: Critical paths, family of $h^m$ heuristics
Heuristics: Recap

Coarse overview on common heuristics for planning

- Central paradigm: Relaxation
- Heuristics based on Delete-relaxation:
  - Easy heuristics measuring goal distance
  - Fast-Forward heuristic $h^{FF}$
  - Family of Landmark heuristics
- Relaxation by abstraction: Heuristics on pattern databases
- Relaxation by parallelism: Critical paths, family of $h^m$ heuristics

More resources and information:

- *Automated Planning and Acting* [GNT16]
- *A Beginner’s Introduction to Heuristic Search Planning*: Online slides [HR15]
- Don’t be afraid to read some referenced papers!
Heuristics: I can see a pattern there . . .

- Admissible and accurate heuristics can be very expensive
  - Example: Abstraction heuristics / Pattern databases
- Admissible and inexpensive heuristics are often inaccurate
  - Example: $h_d^+(s)$
- Accurate and inexpensive heuristics are hardly admissible
  - Example: “original” $h^{FF}(s)$ with approximated action sets
Heuristics: I can see a pattern there . . .

- **Admissible and accurate** heuristics can be very expensive
  - Example: Abstraction heuristics / Pattern databases

- **Admissible and inexpensive** heuristics are often inaccurate
  - Example: \( h_d^+ (s) \)

- **Accurate and inexpensive** heuristics are hardly admissible
  - Example: “original” \( h^{FF} (s) \) with approximated action sets

- **Is this truly accurate?**
  - Possible: Accurate, polynomial complexity, “almost admissible”
  - Contained problems like Hitting Set can be **easy in practice** (small instances)
Issues with empirical heuristics

*Using the benchmarks for inspiration during development, we have been able to come up with a heuristic method that is not probably efficient, but does work well empirically on a large class of planning tasks.* — J. Hoffmann et al., 2001 [HN01]

- Heuristics are driven by benchmark instances
- Benchmark instances are designed using current (heuristic) planners
Issues with empirical heuristics

Using the benchmarks for inspiration during development, we have been able to come up with a heuristic method that is not probably efficient, but does work well empirically on a large class of planning tasks. — J. Hoffmann et al., 2001 [HN01]

- Heuristics are driven by benchmark instances
- Benchmark instances are designed using current (heuristic) planners
  ⇒ Feedback loop: Can lead to overfitting effects
    - Heuristics will perform well on considered problems
    - Even small adjustments in some problems may have “unjustifiable” performance impact
Issues with empirical heuristics

Using the benchmarks for inspiration during development, we have been able to come up with a heuristic method that is not probably efficient, but does work well empirically on a large class of planning tasks. — J. Hoffmann et al., 2001 [HN01]

- Heuristics are driven by benchmark instances
- Benchmark instances are designed using current (heuristic) planners
  - Feedback loop: Can lead to overfitting effects
    - Heuristics will perform well on considered problems
    - Even small adjustments in some problems may have "unjustifiable" performance impact
- How to avoid overfitting?
  - Try many different planning domains and problems
  - Have some (theoretical) justification for why the heuristic performs good (or bad) on problems
The portfolio approach

- Many heuristics are good on some planning domains, bad on others
- Take this as an opportunity: Why not use all of them?
The portfolio approach

- Many heuristics are good on some planning domains, bad on others
- Take this as an opportunity: Why not use all of them?
- Automated Planning with a portfolio
  - Initialize multiple planners searching the same problem, with different heuristics
  - Exploit multi-core architectures to let all planners run in parallel
  - “First one wins” (or: later solutions may improve plan)
The portfolio approach

- Many heuristics are good on some planning domains, bad on others
- Take this as an opportunity: Why not use all of them?
- Automated Planning with a portfolio
  - Initialize multiple planners searching the same problem, with different heuristics
  - Exploit multi-core architectures to let all planners run in parallel
  - “First one wins” (or: later solutions may improve plan)
- Pros:
  - Allows “blind” use of specialized heuristics for general problems
  - Easy way to exploit parallelism
The portfolio approach

- Many heuristics are good on some planning domains, bad on others
- Take this as an opportunity: Why not use all of them?
- Automated Planning with a portfolio
  - Initialize multiple planners searching the same problem, with different heuristics
  - Exploit multi-core architectures to let all planners run in parallel
  - “First one wins” (or: later solutions may improve plan)
- Pros:
  - Allows “blind” use of specialized heuristics for general problems
  - Easy way to exploit parallelism
- Cons:
  - Space use linear in amount of workers (if done naively)
  - May scale badly with a lot of cores
Stay tuned!

Next lecture: Plan Space Search, and the GraphPlan algorithm
References I


