Automated Planning Tools for Intelligent Decision Making

3. AI Planning, Part II: Planners and Extensions

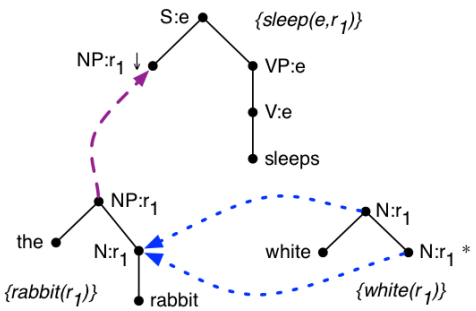
Álvaro Torralba

AALBORG UNIVERSITET

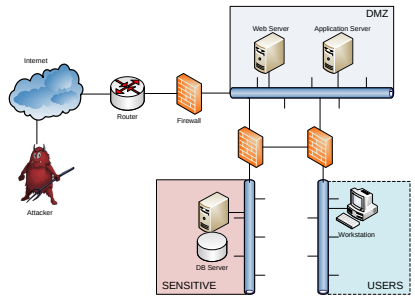
Spring 2021

Agenda

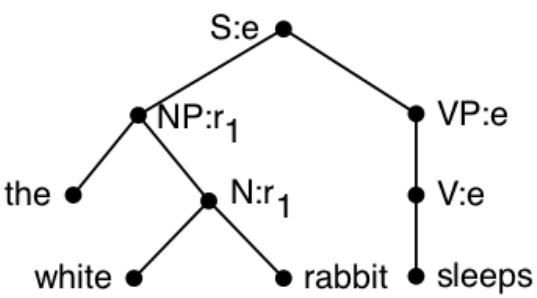
Planning

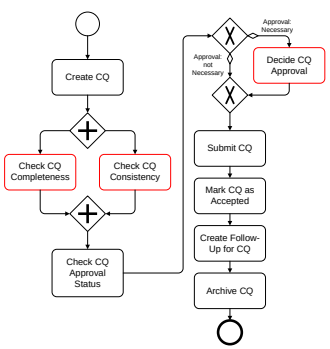


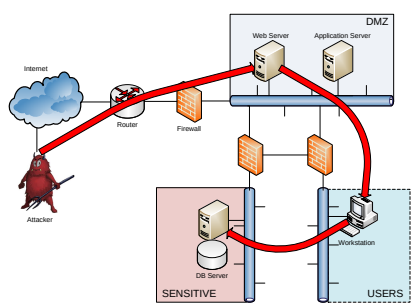
Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived



Planning Domain Definition Language (PDDL) $\mapsto$ Planning System







STRIPS Planning: Syntax

Definition (STRIPS Planning Task). *A STRIPS planning task, short planning task, is a 4-tuple $\Pi = (P, A, I, G)$ where:*

- P *is a finite set of facts (aka propositions).*
- A *is a finite set of actions; each* $a \in A$ *is a triple* $a = (pre_a, add_a, del_a)$ *of subsets of* P *referred to as the action's precondition, add list, and delete list respectively; we require that* $add_a \cap del_a = \emptyset$.
- $I \subseteq P$ *is the initial state.*
- $G \subseteq P$ *is the goal.*

We will often give each action $a \in A$ *a name (a string), and identify* a *with that name.*

STRIPS Planning: Syntax

Definition (STRIPS Planning Task). *A STRIPS planning task, short planning task, is a 4-tuple $\Pi = (P, A, I, G)$ where:*

- *P is a finite set of facts (aka propositions).*
- *A is a finite set of actions; each $a \in A$ is a triple $a = (pre_a, add_a, del_a)$ of subsets of P referred to as the action's precondition, add list, and delete list respectively; we require that $add_a \cap del_a = \emptyset$.*
- *$I \subseteq P$ is the initial state.*
- *$G \subseteq P$ is the goal.*

We will often give each action $a \in A$ a name (a string), and identify a with that name.

Note: We assume unit costs for simplicity: every action has cost 1.

PDDL

- Action-centric language:
  - Preconditions: when can actions be executed
  - Effects: how actions affect the world

Open Discussion

- Homework: Round of Discussion

Open Discussion

- Homework: Round of Discussion
- What are your first impressions? UPAAL, ILP, PDDL

The International Planning Competition (IPC)

Competition?

"Run competing planners on a set of benchmarks devised by the IPC organizers. Give awards to the most effective planners."

The International Planning Competition (IPC)

Competition?

"Run competing planners on a set of benchmarks devised by the IPC organizers. Give awards to the most effective planners."

- 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018
- PDDL [McDermott *et al.* (1998); Fox and Long (2003); Hoffmann and Edelkamp (2005); Gerevini *et al.* (2009)]
- ≈ 70 domains, > 1500 instances, 74 planning systems in 2011
- Optimal track vs. satisficing track
- Various others: uncertainty, learning, ...

`http://ipc.icaps-conference.org/`

PDDL Support

List of PDDL features (sorted by how well are they supported by current planners):

- STRIPS, Types, Negative preconditions, Action cost
  $\rightarrow$(Almost) complete support

PDDL Support

List of PDDL features (sorted by how well are they supported by current planners):

- STRIPS, Types, Negative preconditions, Action cost
  $\rightarrow$(Almost) complete support
- Conditional Effects (when), Quantified Effects (forall)
  $\rightarrow$Common though sometimes may be detrimental for performance

PDDL Support

List of PDDL features (sorted by how well are they supported by current planners):

- STRIPS, Types, Negative preconditions, Action cost
  $\rightarrow$(Almost) complete support
- Conditional Effects (when), Quantified Effects (forall)
  $\rightarrow$Common though sometimes may be detrimental for performance
- Quantified preconditions (forall, exists), Derived Predicates
  $\rightarrow$Only a subset of planners support them

PDDL Support

List of PDDL features (sorted by how well are they supported by current planners):

- STRIPS, Types, Negative preconditions, Action cost
  $\rightarrow$(Almost) complete support
- Conditional Effects (when), Quantified Effects (forall)
  $\rightarrow$Common though sometimes may be detrimental for performance
- Quantified preconditions (forall, exists), Derived Predicates
  $\rightarrow$Only a subset of planners support them

Modelling Advice: Use the simpler model possible

How (most) planners work?

Why? Is this relevant for the user?

How (most) planners work?

Why? Is this relevant for the user? Yes, understanding the basic ideas behind planning algorithms helps for writing useful models
$\rightarrow$When the planner fails, you can diagnose it and improve the model or find a more suitable planner

How (most) planners work?

Why? Is this relevant for the user? Yes, understanding the basic ideas behind planning algorithms helps for writing useful models
$\rightarrow$When the planner fails, you can diagnose it and improve the model or find a more suitable planner

Two phases:

- Preprocessing (Grounding)
- Search

How (most) planners work?

Why? Is this relevant for the user? Yes, understanding the basic ideas behind planning algorithms helps for writing useful models
→When the planner fails, you can diagnose it and improve the model or find a more suitable planner

Two phases:

- Preprocessing (Grounding)
  →Transforms the PDDL input to an internal representation (close to STRIPS)
- Search

How (most) planners work?

Why? Is this relevant for the user? Yes, understanding the basic ideas behind planning algorithms helps for writing useful models
→When the planner fails, you can diagnose it and improve the model or find a more suitable planner

Two phases:

- Preprocessing (Grounding)
  →Transforms the PDDL input to an internal representation (close to STRIPS)
- Search
  →Finds an (optimal) plan

Grounding

PDDL:

```
(:predicates
   (at ?t - truck ?l - loc)
)

(:action drive
   :parameters  (?t - truck ?l1 - loc ?l2 - loc)
   ...
)
```

Grounding

PDDL:

```
(:predicates
   (at ?t - truck ?l - loc)
)

(:action drive
   :parameters  (?t - truck ?l1 - loc ?l2 - loc)
   ...
)
```

Grounded Representation (Strips):

```
(at truck1 Aalborg) (at truck1 Aarhus) (at truck1 Copenhagen)
(at truck2 Aalborg) (at truck2 Aarhus) (at truck2 Copenhagen)
...

(drive truck1 Aalborg Aarhus) (drive truck1 Aarhus Aalborg)
(drive truck1 Aarhus Copenhagen) ...
```

Grounding

PDDL:

```
(:predicates
   (at ?t - truck ?l - loc)
)

(:action drive
   :parameters  (?t - truck ?l1 - loc ?l2 - loc)
   ...
)
```

Grounded Representation (Strips):

```
(at truck1 Aalborg) (at truck1 Aarhus) (at truck1 Copenhagen)
(at truck2 Aalborg) (at truck2 Aarhus) (at truck2 Copenhagen)
...

(drive truck1 Aalborg Aarhus) (drive truck1 Aarhus Aalborg)
(drive truck1 Aarhus Copenhagen) ...
```

→Size of the grounded representation is much bigger than the original PDDL (exponential in the number of parameters of predicates and action schemas)

Grounding: Finite-domain variables

(at truck1 Aalborg) (at truck1 Aarhus) (at truck1 Copenhagen)

What have these three facts in common?

Grounding: Finite-domain variables

(at truck1 Aalborg) (at truck1 Aarhus) (at truck1 Copenhagen)

What have these three facts in common?

→Exactly one of them will be true in every reachable state!

Grounding: Finite-domain variables

(at truck1 Aalborg) (at truck1 Aarhus) (at truck1 Copenhagen)

What have these three facts in common?

→Exactly one of them will be true in every reachable state!

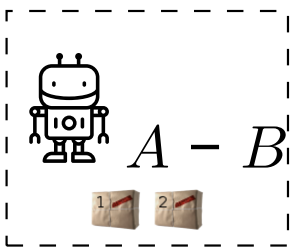
Most planners use finite-domain variables internally (instead of only Boolean).

Example: Variable at-truck1 has 3 values: Aalborg, Aarhus, Copenhagen

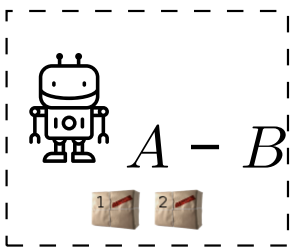
History of Planning Algorithms

- Compilation into Logics/Theorem Proving: (popular: Stone Age – 1990)
- Partial-Order Planning: (popular: 1990 – 1995)
- GraphPlan: (popular 1995 – 2000)
- Planning as SAT: (popular 1996 – today)
- Planning as Heuristic Search: (popular 1996 – today)
- Planning as Symbolic Search (with BDDs): (popular 2000 – today)

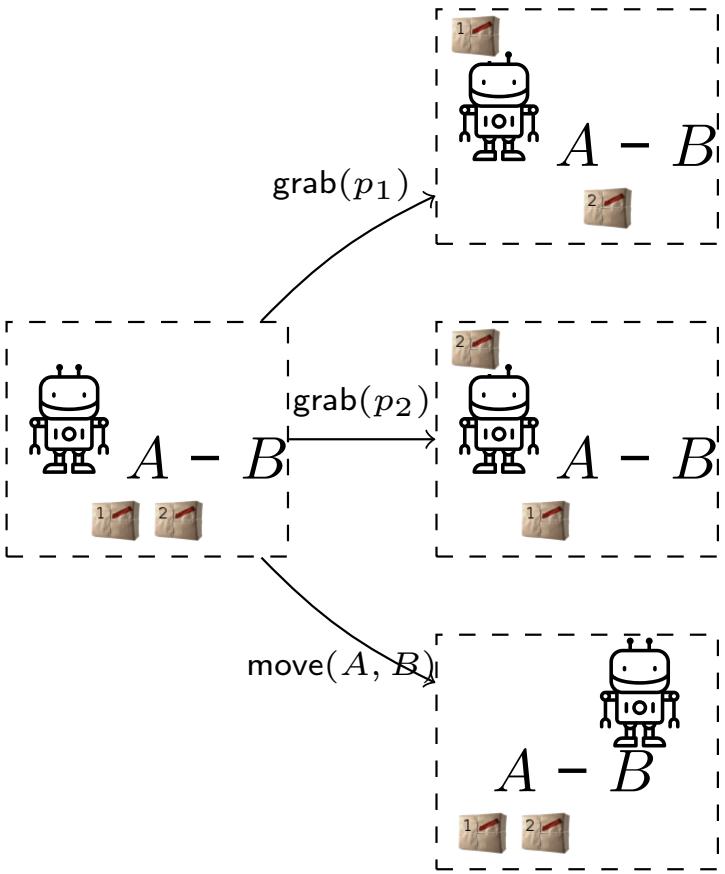
How to solve planning tasks? Search!



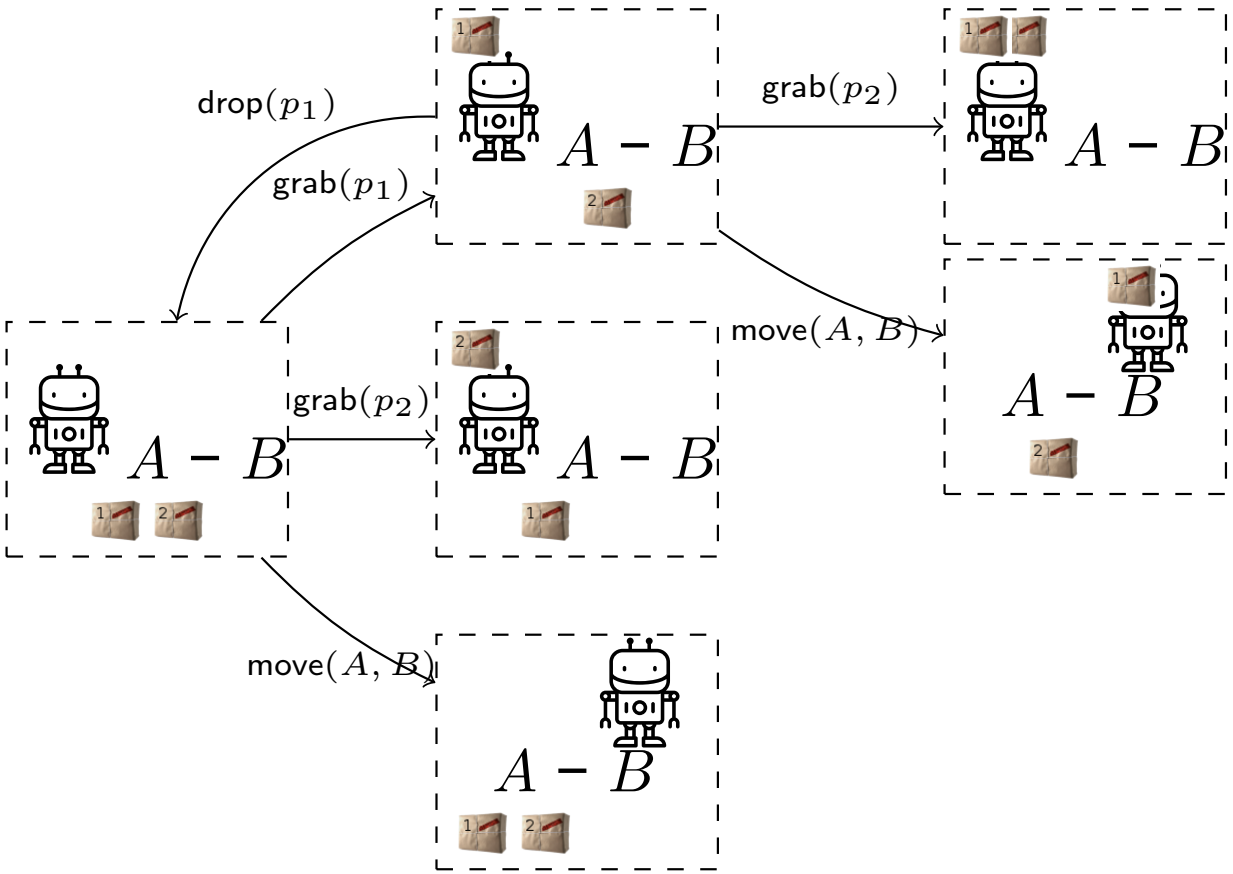
How to solve planning tasks? Search!



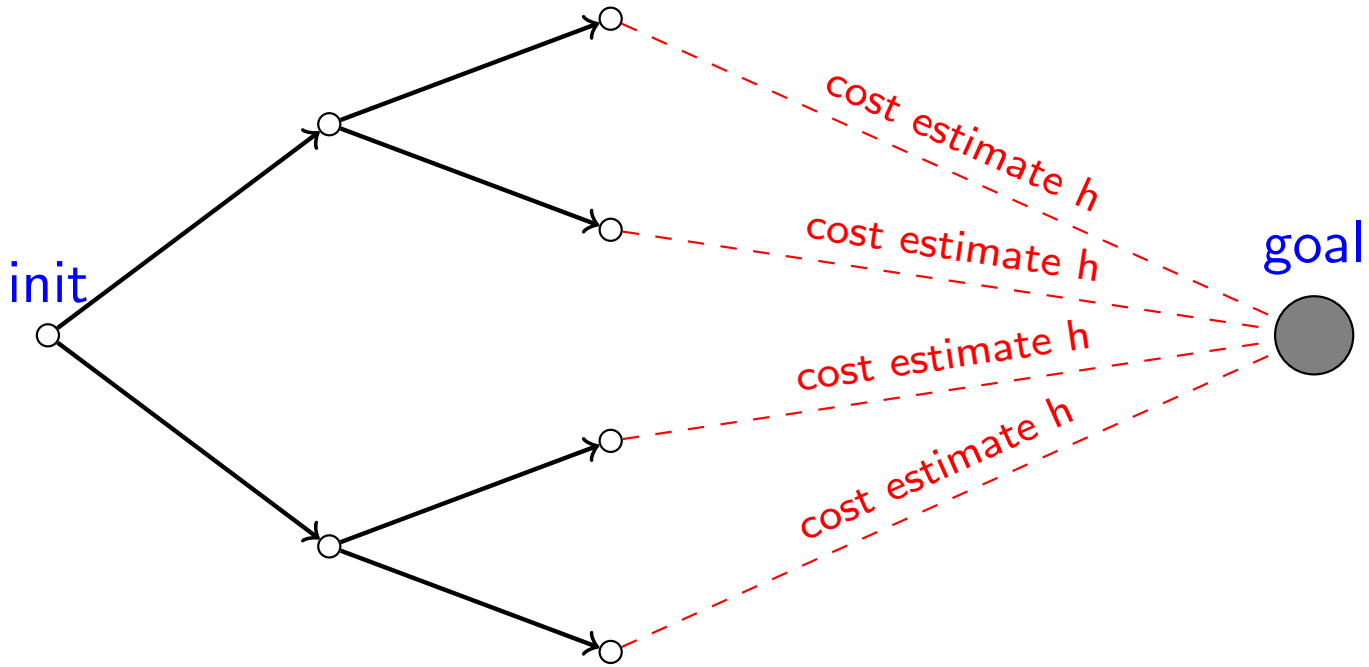
How to solve planning tasks? Search!



How to solve planning tasks? Search!

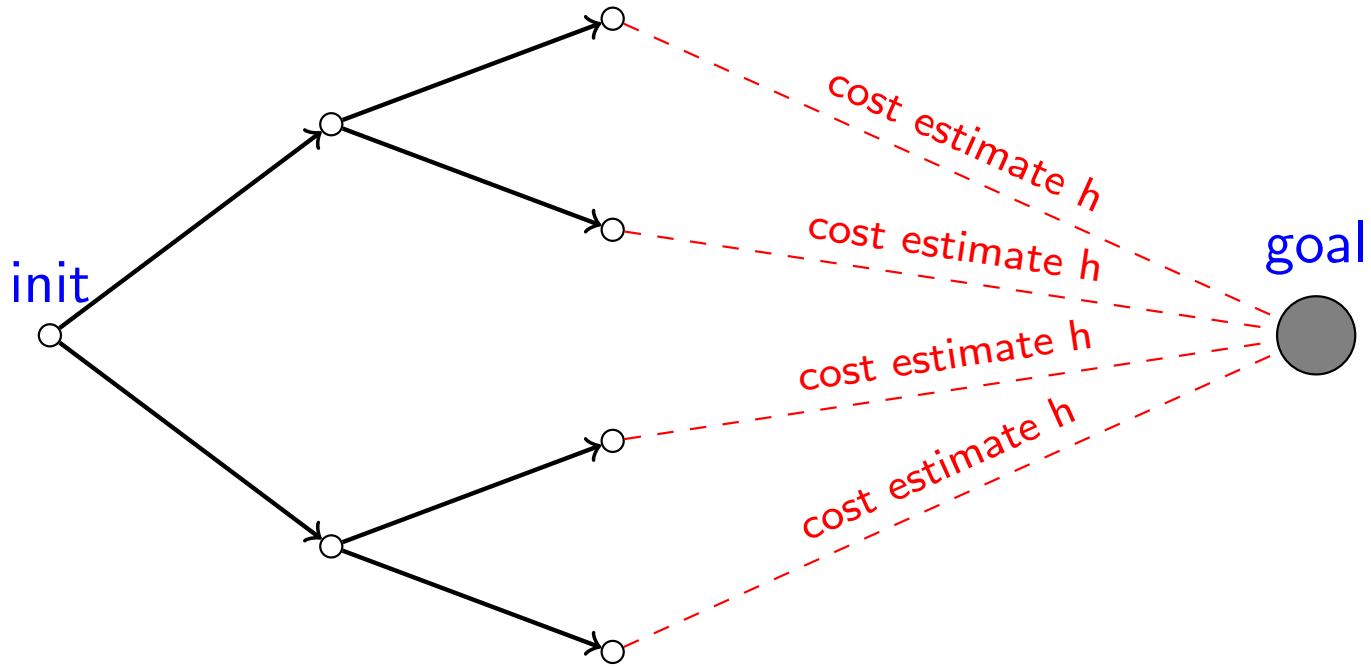


Heuristic Search



$\rightarrow$ Heuristic function h estimates the cost of an optimal path from a state s to the goal; search prefers to expand states s with small $h(s)$.

Heuristic Search



$\rightarrow$ Heuristic function h estimates the cost of an optimal path from a state s to the goal; search prefers to expand states s with small $h(s)$.

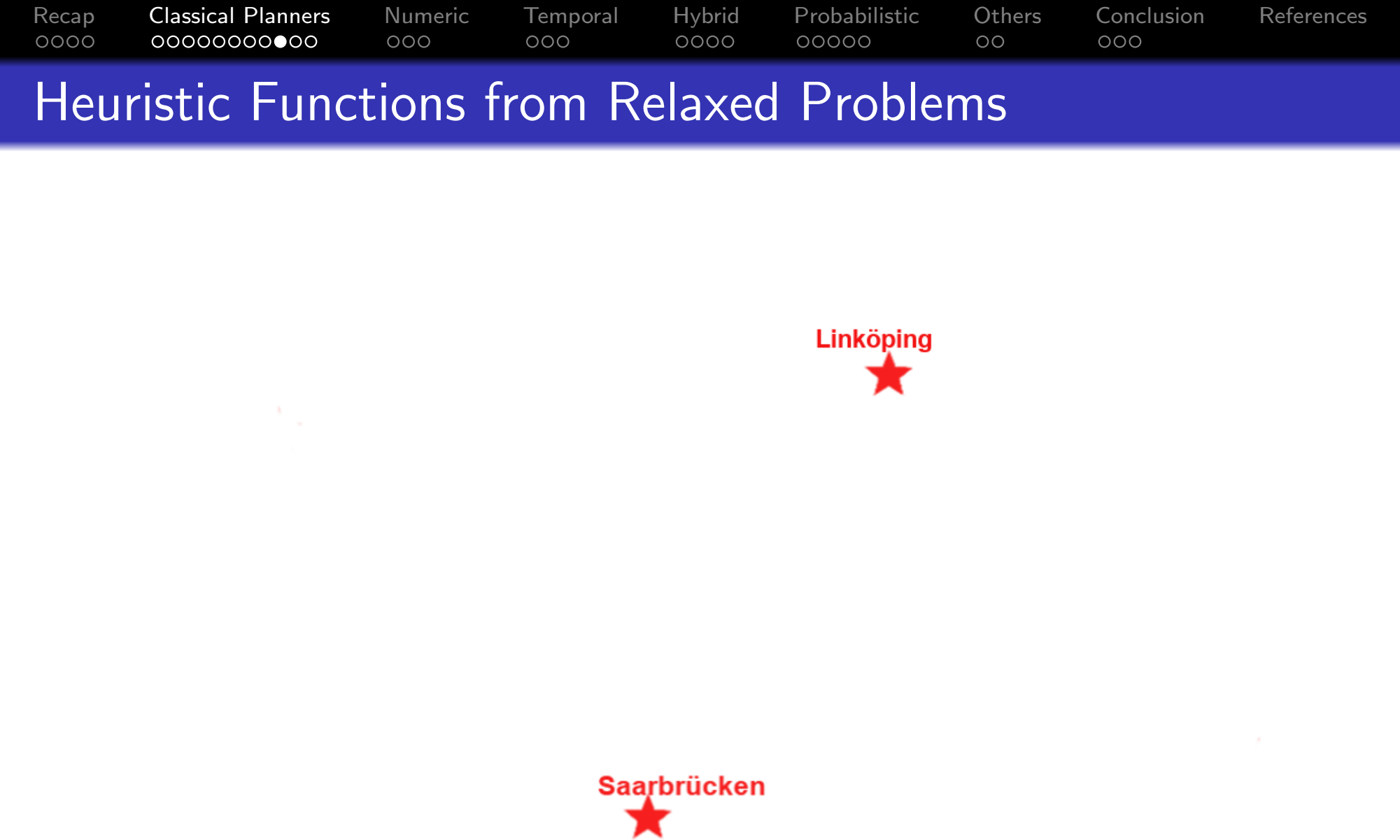
Live Demo vs. Breadth-First Search:

`http://qiao.github.io/PathFinding.js/visual/`

Heuristic Functions from Relaxed Problems

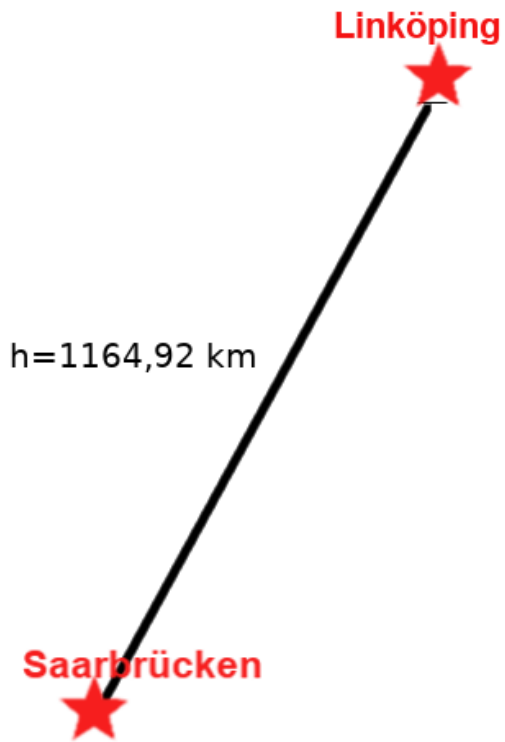
Problem II: Find a route from Saarbruecken To Edinburgh.

Heuristic Functions from Relaxed Problems



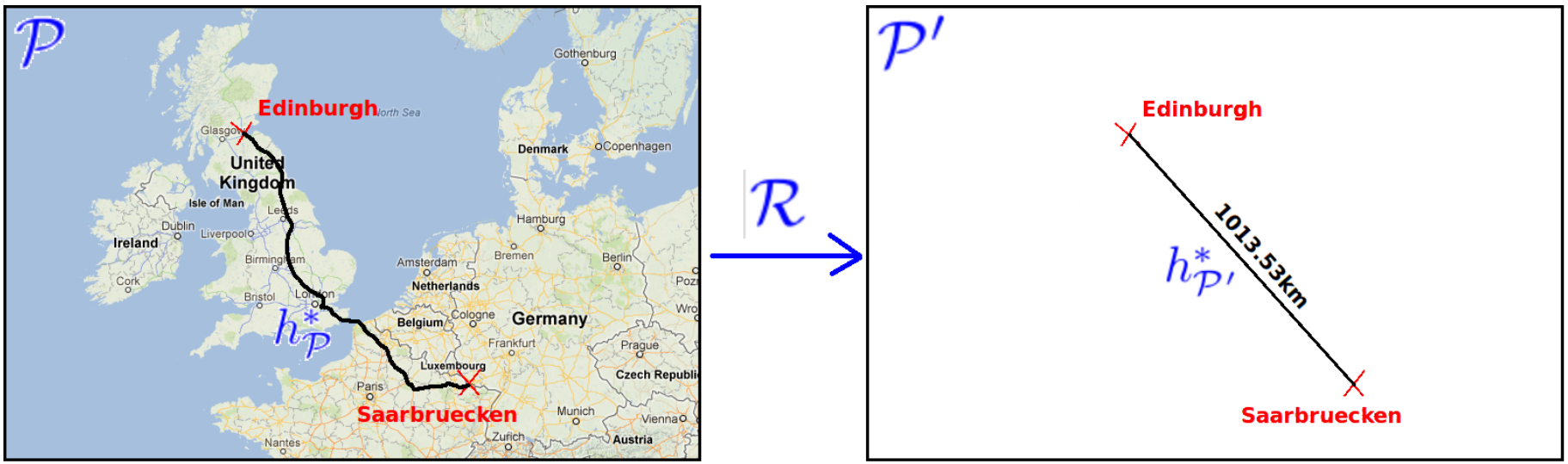
Relaxed Problem Π': Throw away the map.

Heuristic Functions from Relaxed Problems



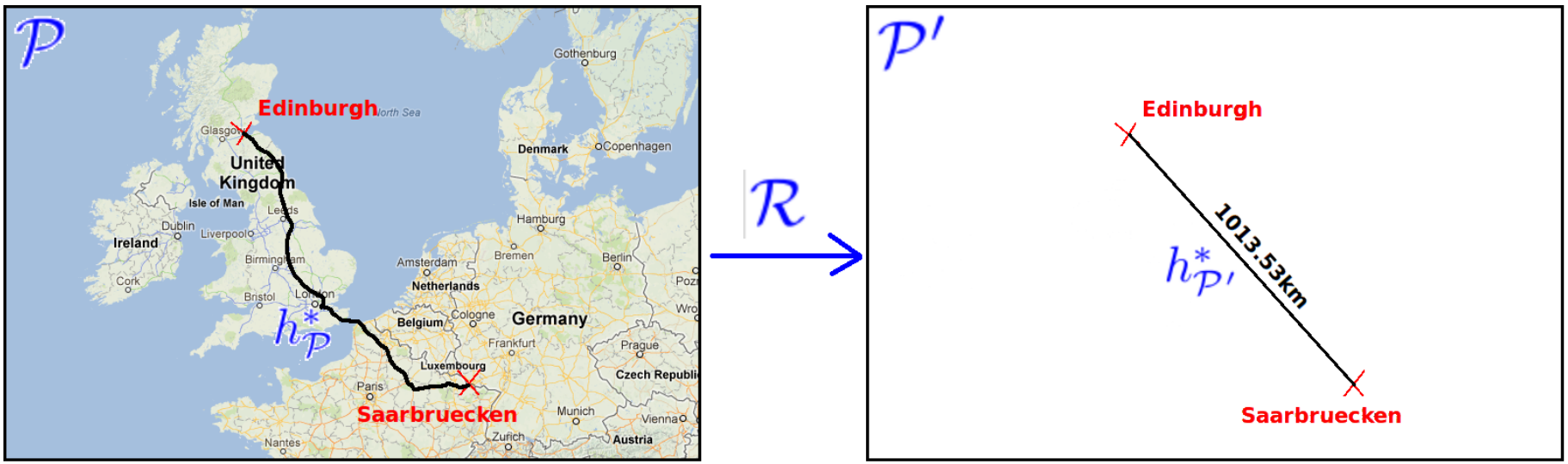
Heuristic function h: Straight line distance.

Relaxation in Route-Finding



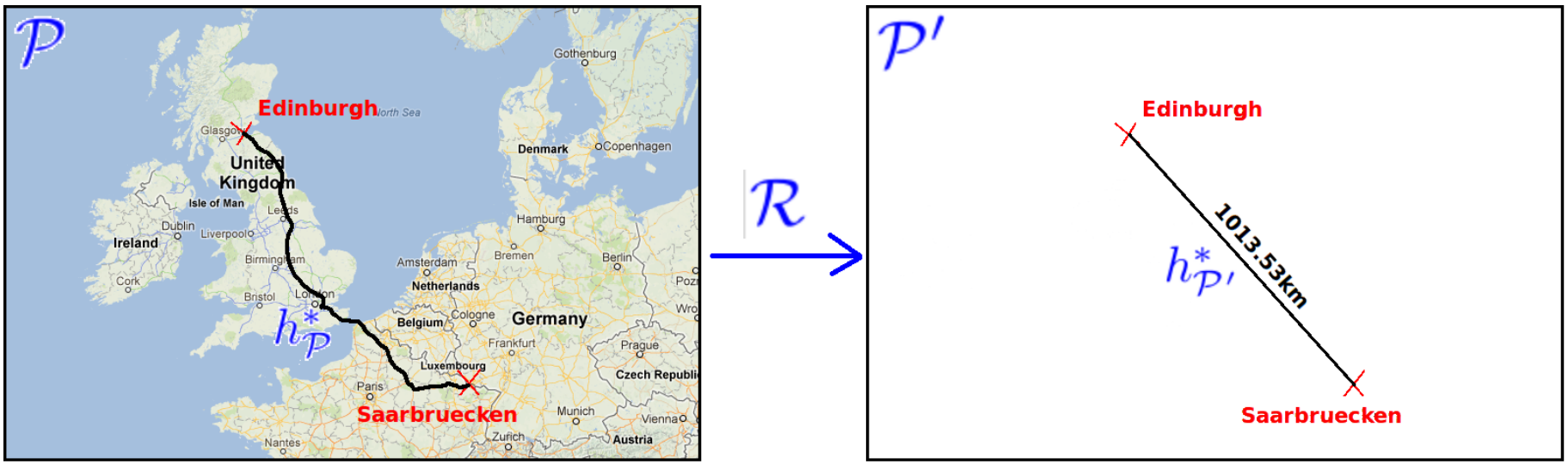
- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.

Relaxation in Route-Finding



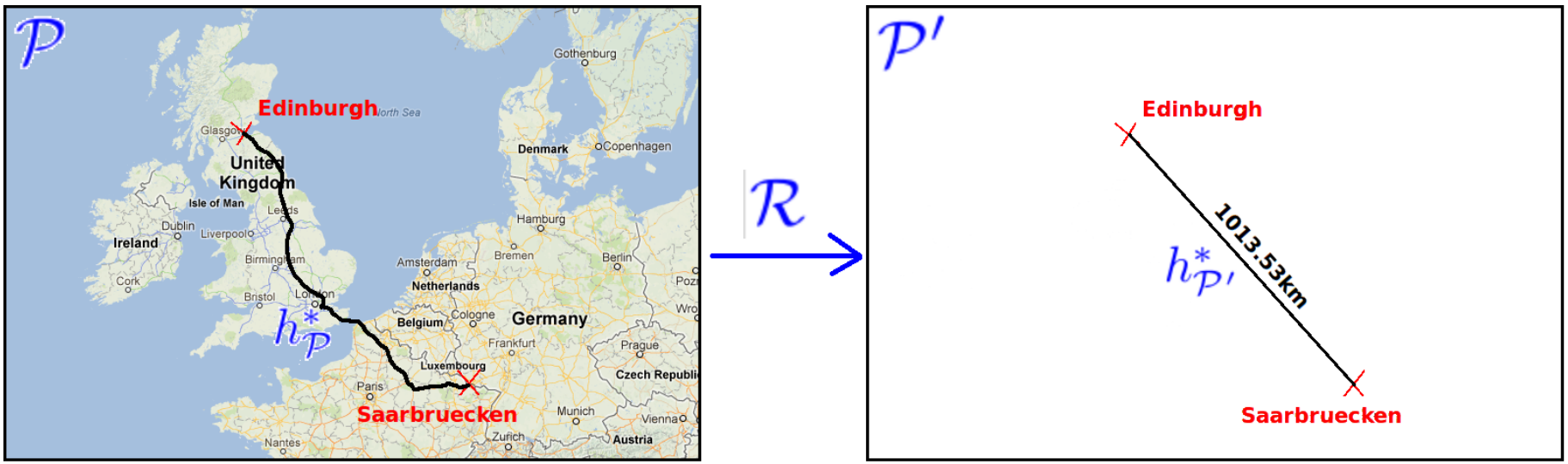
- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$:

Relaxation in Route-Finding



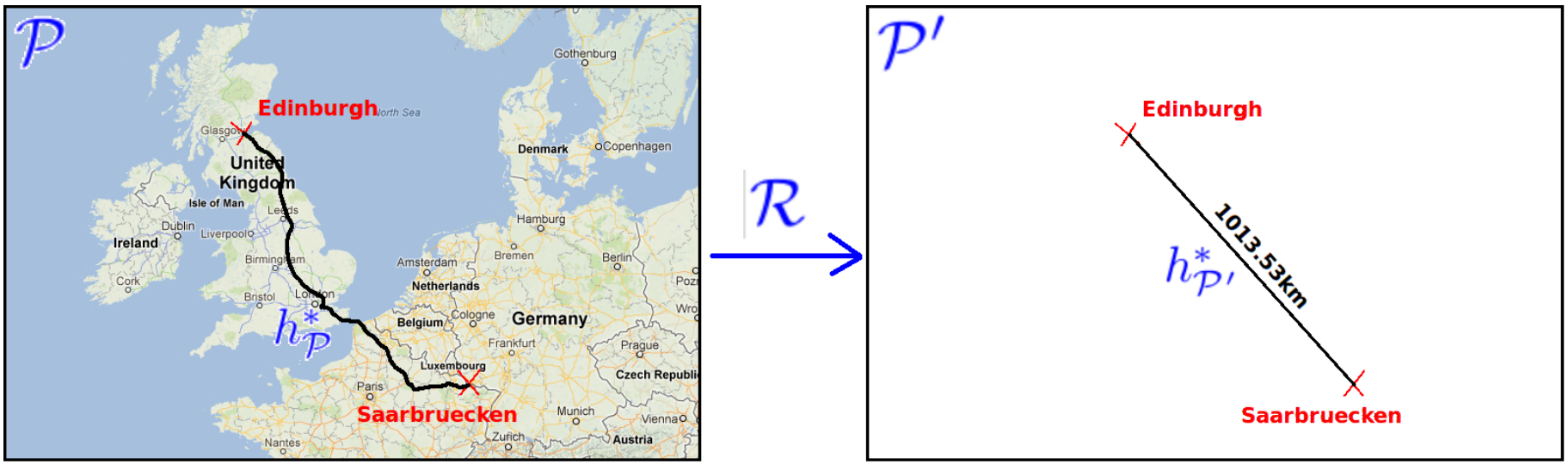
- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$: Route finding on an empty map.
- Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$:

Relaxation in Route-Finding



- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$: Route finding on an empty map.
- Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$: Straight-line distance.
- Transformation $\mathcal{R}$:

Relaxation in Route-Finding



- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$: Route finding on an empty map.
- Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$: Straight-line distance.
- Transformation $\mathcal{R}$: Throw away the map.

Planning Heuristics

Challenge: Given a planning task Π, simplify Π to obtain a relaxed problem Π', then solve Π' to obtain the heuristic estimate h. All of this must be fully automatic.

Planning Heuristics

Challenge: Given a planning task Π, simplify Π to obtain a relaxed problem Π', then solve Π' to obtain the heuristic estimate h. All of this must be fully automatic.

This is a HUGE playground! Abstract/relax the world WHICHEVER way!

Planning Heuristics

Challenge: Given a planning task Π, simplify Π to obtain a relaxed problem Π', then solve Π' to obtain the heuristic estimate h. All of this must be fully automatic.

This is a HUGE playground! Abstract/relax the world WHICHEVER way!

1. Delete-relaxation

   $\rightarrow$Ignore the negative effects of the actions

Planning Heuristics

Challenge: Given a planning task Π, simplify Π to obtain a relaxed problem Π', then solve Π' to obtain the heuristic estimate h. All of this must be fully automatic.

This is a HUGE playground! Abstract/relax the world WHICHEVER way!

1. Delete-relaxation

   $\rightarrow$Ignore the negative effects of the actions

2. Abstractions

   $\rightarrow$Look only at part of the task

Planning Heuristics

Challenge: Given a planning task Π, simplify Π to obtain a relaxed problem Π', then solve Π' to obtain the heuristic estimate h. All of this must be fully automatic.

This is a HUGE playground! Abstract/relax the world WHICHEVER way!

1. Delete-relaxation
   $\rightarrow$Ignore the negative effects of the actions
2. Abstractions
   $\rightarrow$Look only at part of the task
3. Landmarks
   $\rightarrow$What facts need to be achieved on our way to the goal?

Planning Heuristics

Challenge: Given a planning task Π, simplify Π to obtain a relaxed problem Π', then solve Π' to obtain the heuristic estimate h. All of this must be fully automatic.

This is a HUGE playground! Abstract/relax the world WHICHEVER way!

1. Delete-relaxation
   $\rightarrow$Ignore the negative effects of the actions
2. Abstractions
   $\rightarrow$Look only at part of the task
3. Landmarks
   $\rightarrow$What facts need to be achieved on our way to the goal?
4. Critical Paths
   $\rightarrow$Decompose states into subsets of K facts

Planning Heuristics

Challenge: Given a planning task Π, simplify Π to obtain a relaxed problem Π', then solve Π' to obtain the heuristic estimate h. All of this must be fully automatic.

This is a HUGE playground! Abstract/relax the world WHICHEVER way!

1. Delete-relaxation
   $\rightarrow$Ignore the negative effects of the actions
2. Abstractions
   $\rightarrow$Look only at part of the task
3. Landmarks
   $\rightarrow$What facts need to be achieved on our way to the goal?
4. Critical Paths
   $\rightarrow$Decompose states into subsets of K facts
5. Operator Counting
   $\rightarrow$Use LP methods to estimate how many actions need to be applied

Numeric Planning (PDDL 2.1)

In the real world: We have numbers!

Numeric STRIPS Planning: Extends STRIPS by introducing numeric variables V_n with rational values in $\mathbb{Q}$.

- Numeric expressions: We can do simple arithmetic $(+, -, \times, \div)$ with the values of variables and/or constants.

Numeric Planning (PDDL 2.1)

In the real world: We have numbers!

Numeric STRIPS Planning: Extends STRIPS by introducing numeric variables V_n with rational values in $\mathbb{Q}$.

- Numeric expressions: We can do simple arithmetic $(+, -, \times, \div)$ with the values of variables and/or constants.
- Numeric conditions: compare numeric expressions with $\{<, \leq, =, \geq, >\}$.

Numeric Planning (PDDL 2.1)

In the real world: We have numbers!

Numeric STRIPS Planning: Extends STRIPS by introducing numeric variables V_n with rational values in $\mathbb{Q}$.

- Numeric expressions: We can do simple arithmetic $(+, -, \times, \div)$ with the values of variables and/or constants.
- Numeric conditions: compare numeric expressions with $\{<, \leq, =, \geq, >\}$.
- Numeric effects: assign a numeric expression to a numeric variable in V_n.

Numeric Planning (PDDL 2.1)

In the real world: We have numbers!

Numeric STRIPS Planning: Extends STRIPS by introducing numeric variables V_n with rational values in $\mathbb{Q}$.

- Numeric expressions: We can do simple arithmetic $(+, -, \times, \div)$ with the values of variables and/or constants.
- Numeric conditions: compare numeric expressions with $\{<, \leq, =, \geq, >\}$.
- Numeric effects: assign a numeric expression to a numeric variable in V_n.

ENHSP: (`https://sites.google.com/view/enhsp/`)

(examples in the next slides were taken from the ENHSP webpage)

Simple Linear Numeric Planning (Example)

```
(define (domain fn-counters)
    ;(:requirements :strips :typing :equality :adl)
    (:types counter)

    (:functions
        (value ?c - counter);; - int   ;; The value shown in counter ?
        (max_int);; -  int ;; The maximum integer we consider - a sta
    )

    ;; Increment the value in the given counter by one
    (:action increment
        :parameters (?c - counter)
        :precondition (and (<= (+ (value ?c) 1) (max_int)))
        :effect (and (increase (value ?c) 1))
    )

    ;; Decrement the value in the given counter by one
    (:action decrement
        :parameters (?c - counter)
        :precondition (and (>= (value ?c) 1))
        :effect (and (decrease (value ?c) 1))
    )
)
```

Simple Linear Numeric Planning (Example)

```
(define (problem instance_4)
  (:domain fn-counters)
  (:objects
    c0 c1 c2 c3 - counter
  )

  (:init
    (= (max_int) 8)
        (= (value c0) 0)
        (= (value c1) 0)
        (= (value c2) 0)
        (= (value c3) 0)
  )

  (:goal (and
    (<= (+ (value c0) 1) (value c1))
        (<= (+ (value c1) 1) (value c2))
        (<= (+ (value c2) 1) (value c3))
  ))
)
```

Temporal Planning (PDDL 2.1)

In the real world: Events do not happen instantaneously!

Temporal Planning (PDDL 2.1)

In the real world: Events do not happen instantaneously!

In classical planning the action effects happen immediately. However, in the real world, actions take time to execute.
When the precondition needs to hold? When the effect is applied?

Temporal Planning (PDDL 2.1)

In the real world: Events do not happen instantaneously!

In classical planning the action effects happen immediately. However, in the real world, actions take time to execute.
When the precondition needs to hold? When the effect is applied?

- preconditions:
- effects:

Temporal Planning (PDDL 2.1)

In the real world: Events do not happen instantaneously!

In classical planning the action effects happen immediately. However, in the real world, actions take time to execute.
When the precondition needs to hold? When the effect is applied?

- preconditions: at-start
- effects: at-end

Temporal Planning (PDDL 2.1)

In the real world: Events do not happen instantaneously!

In classical planning the action effects happen immediately. However, in the real world, actions take time to execute.
When the precondition needs to hold? When the effect is applied?

- preconditions: at-start , at-end
- effects: at-start, at-end

Temporal Planning (PDDL 2.1)

In the real world: Events do not happen instantaneously!

In classical planning the action effects happen immediately. However, in the real world, actions take time to execute.
When the precondition needs to hold? When the effect is applied?

- preconditions: at-start , at-end , over-all
- effects: at-start, at-end , over-all (continuous change)

Temporal Planning (PDDL 2.1)

In the real world: Events do not happen instantaneously!

In classical planning the action effects happen immediately. However, in the real world, actions take time to execute.
When the precondition needs to hold? When the effect is applied?

- preconditions: at-start , at-end , over-all
- effects: at-start, at-end , over-all (continuous change)

- POPF: `https://nms.kcl.ac.uk/planning/software/popf.html`
- Optic: `https://nms.kcl.ac.uk/planning/software/optic.html`
- IBaCOP: `https://icenamor.github.io/portfolio/Temporal/`

Temporal Planning (Example Fox and Long (2003))

```
(:durative-action heat-water
   :parameters (?p - pan)

   :duration (= ?duration (/ (- 100 (temperature ?p)) (heat-rate)))

   :condition (and (at start (full ?p))
                   (at start (onHeatSource ?p))
                   (at start (byPan))
                   (over all (full ?p))
                   (over all (onHeatSource ?p))
                   (over all (heating ?p))
                   (at end (byPan)))

   :effect (and (at start (heating ?p))
                (at end (not (heating ?p)))
                (at end (assign (temperature ?p) 100)))
           )
```

Temporal Planning (Example Fox and Long (2003))

We can also encode continuous effects:

```
(:durative-action heat-water
   :parameters (?p - pan)

   :duration ()

   :condition (and (at start (full ?p))
                   (at start (onHeatSource ?p))
                   (at start (byPan))
                   (over all (full ?p))
                   (over all (onHeatSource ?p))
                   (over all (heating ?p))
                   (over all (<= (temperature ?p) 100))
                   (at end (byPan)))

    :effect (and (at start (heating ?p))
                 (at end (not (heating ?p)))
                  (increase (temperature ?p) (* #t (heat-rate))))
)
```

Hybrid Planning (PDDL +)

In the real world: The state of the world changes independently of our actions!

Hybrid Planning (PDDL +)

In the real world: The state of the world changes independently of our actions!

- Exogenous events: Happen instantaneously (discrete)
  →Example: Someone/something hits you, changing your direction

Hybrid Planning (PDDL +)

In the real world: The state of the world changes independently of our actions!

- Exogenous events: Happen instantaneously (discrete)
  →Example: Someone/something hits you, changing your direction
- Continuous processes: Gradual changes
  →Example: Gravity

Hybrid Planning (PDDL +)

In the real world: The state of the world changes independently of our actions!

- Exogenous events: Happen instantaneously (discrete)
  →Example: Someone/something hits you, changing your direction
- Continuous processes: Gradual changes
  →Example: Gravity

- ENHSP: `https://sites.google.com/view/enhsp/`
- SMTPlan: `http://kcl-planning.github.io/SMTPlan/`
- Dino: `http://kcl-planning.github.io/Dino/`

Example in the next slides from: `http://planning.wiki`

Hybrid Planning (Example)

```
(define (domain car_nonlinear_mt_sc)
 (:predicates (engine_running) (engine_stopped) )
 (:functions
    (d) (v) (a) (drag_coefficient) (max_speed)
    (max_acceleration) (min_acceleration) )

 (:constraint speed_limit
  :condition (and (>= (v) (* -1 (max_speed))) (<= (v) (max_speed))))

 (:process displacement
   :precondition (and (engine_running) (> (v) 0))
   :effect (increase (d) (* #t (v))))

 (:process moving_drag
  :precondition (engine_running)
   :effect (increase (v) (* #t (a)) )  ;; acceleration  )

 (:process drag_ahead
    :precondition (and (engine_running) (> (v) 0))
    :effect (decrease (v) (* #t (* (^ (v) 2) (drag_coefficient)
) ) ) ) )
```

Hybrid Planning (Example)

```
(:action accelerate
    :precondition (and (< (a) (max_acceleration)) (engine_running) )
    :effect (increase (a) 1.0)
)

(:action stop_car
 :precondition (and (> (v) -0.1) (< (v) 0.1) (= (a) 0.0) (engine_run
 :effect (and  (assign (v) 0.0)
               (engine_stopped)
               (not (engine_running))) )

(:action start_car
    :precondition (engine_stopped)
    :effect (and (engine_running)
                 (not (engine_stopped))))

(:action decelerate
    :precondition (and (> (a) (min_acceleration)) (engine_running))
    :effect (decrease (a) 1.0)
)
)
```

Hybrid Planning (Example)

```
(define (problem instance_1_300_01_100)
  (:domain car_nonlinear_mt_sc)

  (:init
    (= (d) 0.0)
       (= (v) 0.0)
       (engine_stopped)
       (= (a) 0.0)
       (= (max_acceleration) 1)
       (= (min_acceleration) -1)
       (= (drag_coefficient) 0.1)
       (= (max_speed) 10.0)
  )

  (:goal
    (and
       (>= (d) 29.5 )
       (<= (d) 30.5 )
       (engine_stopped)
       )
  )
)
```

Non-deterministic/Stochastic Environments

In the real world: we cannot always anticipate the effect of our actions!

Differences with respect to classical planning:

Non-deterministic/Stochastic Environments

In the real world: we cannot always anticipate the effect of our actions!

Differences with respect to classical planning:

1. Actions can have multiple outcomes
2. (Optionally) If the probability of each outcome is known, then we call it probabilistic planning.

Non-deterministic/Stochastic Environments

In the real world: we cannot always anticipate the effect of our actions!

Differences with respect to classical planning:

1. Actions can have multiple outcomes
2. (Optionally) If the probability of each outcome is known, then we call it probabilistic planning.

Extensions of PDDL for Probabilistic Planning:

- PPDDL (next slide)
- RDDL (more general)

Probabilistic Planners:

- Prost: `https://github.com/prost-planner/prost`
- IPC: `https://ipc2018-probabilistic.bitbucket.io/`

PPDDL Example

```
(:action try-navigate-l1-l2
 :parameters (?hcur ?hnew - horizon-value ?x - rover)
 :precondition
  (and (horizon ?hcur)
       (horizon-decrement ?hcur ?hnew)
       (can_traverse ?x waypoint1 waypoint2)
       (available ?x)
       (at ?x waypoint1)
       (visible waypoint1 waypoint2)
       (road_isunknown road2))
:effect
   (and (not (horizon ?hcur))
        (horizon ?hnew)
        (increase (total-cost) 1)
        (not (road_isunknown road2))
        (probabilistic
           2/10 (and (road_isblocked road2))
           8/10 (and (road_isfree road2)
                     (not (at ?x waypoint1))
                     (at ?x waypoint2))
    ))
)
```

Online vs. Offline Planning

Given a non-deterministic/probabilistic planning task:
Offline Planning: Plan ahead for every possibility.

- Find contingent plan
- Find (optimal) policy

Online vs. Offline Planning

Given a non-deterministic/probabilistic planning task:
Offline Planning: Plan ahead for every possibility.

- Find contingent plan
- Find (optimal) policy

Online Planning: Decide what to do next: spent some time deciding what action to execute, execute it, observe the result and re-plan if necessary.

Fully-Observable Non-Deterministic Planning (FOND Planning)

In the real world: we cannot always anticipate the effect of our actions! (without probabilities)

PRP:
`https://github.com/qumulab/planner-for-relevant-policies`

Planning Under Partial-Observability (POND)

In the real world: we do not know everything about the initial state! (without probabilities)

- Conformant Planning: A sequence of actions that works for all possible initial states
- Contingent Planning: We can have conditions along our plan

Planning Under Partial-Observability (POND)

In the real world: we do not know everything about the initial state! (without probabilities)

- Conformant Planning: A sequence of actions that works for all possible initial states
- Contingent Planning: We can have conditions along our plan

- Conformant/Contingent-FF: `https://fai.cs.uni-saarland.de/hoffmann/cff.html`
- PO-PRP: `https://github.com/qumulab/planner-for-relevant-policies/wiki/PO-PRP`

Preferences/Soft Goals (PDDL 3)

- Hard goals: In classical planning we must achieve all goal facts
- Soft goals: Facts that are desirable, but not a must (each goal has an associated reward)
- Preferences: sometime-after, sometime-before, always-within, hold-during, hold-after

Preferences/Soft Goals (PDDL 3)

- Hard goals: In classical planning we must achieve all goal facts
- Soft goals: Facts that are desirable, but not a must (each goal has an associated reward)
- Preferences: sometime-after, sometime-before, always-within, hold-during, hold-after

Oversubscription Planning: Fixed cost, maximize achieved goals

Net-benefit Planning: Minimize Cost - Reward
$\rightarrow$Compilable to classical planning (give-up action)

Multi-Agent Planning

In the real world: We are not the single and only agent!

Multi-agent planning: Several agents must collaborate to achieve a common goal.

Key: There is some global information known by all agents but each agent has his own private facts, who do not want to share with the rest.

Multi-Agent Planning

In the real world: We are not the single and only agent!

Multi-agent planning: Several agents must collaborate to achieve a common goal.
Key: There is some global information known by all agents but each agent has his own private facts, who do not want to share with the rest.

Multi-Agent STRIPS Planning: A multi-agent STRIPS planning task, is a 5-tuple $\Pi = (P, A, I, G)$ where:

- P is a finite set of facts, divided in private and public facts.
- A is a finite set of actions; each $a \in A$ is a triple of pre, add, and del, divided in private and public actions.
- $I \subseteq P$ is the initial state.
- $G \subseteq P$ is the goal.

Multi-Agent Planning

In the real world: We are not the single and only agent!

Multi-agent planning: Several agents must collaborate to achieve a common goal.
Key: There is some global information known by all agents but each agent has his own private facts, who do not want to share with the rest.

Multi-Agent STRIPS Planning: A multi-agent STRIPS planning task, is a 5-tuple $\Pi = (P, A, I, G)$ where:

- P is a finite set of facts, divided in private and public facts.
- A is a finite set of actions; each $a \in A$ is a triple of pre, add, and del, divided in private and public actions.
- $I \subseteq P$ is the initial state.
- $G \subseteq P$ is the goal.

$\rightarrow$Agents must communicate during the planning process to share information about how they will achieve the goal

Planning in the Real World

Question!

If most real-world environments are not deterministic, not fully observable, not discrete, not single agent, and temporal. What is classical planning good for?

Planning in the Real World

Question!

If most real-world environments are not deterministic, not fully observable, not discrete, not single agent, and temporal. What is classical planning good for?

- The model does not try to simulate the environment, it is just a tool to take good decisions.

Planning in the Real World

Question!

If most real-world environments are not deterministic, not fully observable, not discrete, not single agent, and temporal. What is classical planning good for?

- The model does not try to simulate the environment, it is just a tool to take good decisions.
- Oftentimes, reasoning with a simplified model can still lead to intelligent decisions and solutions are easier to compute than with more complex models.
- Classical planning is a relaxation of the problem so it can be used in heuristics for more complex types of planning.

Planning in the Real World

Question!

If most real-world environments are not deterministic, not fully observable, not discrete, not single agent, and temporal. What is classical planning good for?

- The model does not try to simulate the environment, it is just a tool to take good decisions.
- Oftentimes, reasoning with a simplified model can still lead to intelligent decisions and solutions are easier to compute than with more complex models.
- Classical planning is a relaxation of the problem so it can be used in heuristics for more complex types of planning.

My two cents: Ideally, we should always provide an accurate description of the environment so that the AI simplifies it when necessary. However, automatic simplification methods are not powerful enough in all cases yet.

Summary

- You can use planning tools to solve your problems:
  $\rightarrow$Encode them in PDDL and use any planner
- Classical planning is very effective at solving large problems
- Non-classical planning models reason about more complex environments such as non-deterministic, partially-observable, continuous, temporal, etc.
- Solving these problems by computing a complete offline policy is hard (though many non-classical planners are able to do this satisfactorily in some domains). Many approaches are online, planning to decide the next action by looking into the future but without considering all alternatives.
- Classical planning and heuristic search techniques are still an important ingredient of many approaches that deal with complex environments.

Additional Resources

- Editor: `http://editor.planning.domains/`
- Wiki: `https://planning.wiki/`
- Benchmarks:
  `https://github.com/aibasel/downward-benchmarks`
- Visual Studio Plugin:
  `https://github.com/jan-dolejsi/vscode-pddl`
- Planners:
  - Fast Downward: `http://www.fast-downward.org/`
  - IPC-18: `https://ipc2018-classical.bitbucket.io/#planners`
- Domains as examples:
  - STRIPS: FD'All $\rightarrow$ Blocksworld, Logistics
  - Action costs: IPC'11 $\rightarrow$ Woodworking, Transport
  - Discretized numbers: IPC'11 $\rightarrow$ Nomystery (fuel consumption)
  - Forall-when: IPC'18 $\rightarrow$ Nurikabe
  - Complex ADL: FD'All $\rightarrow$ Miconic-ADL (ID114)

References I

Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

Jörg Hoffmann and Stefan Edelkamp. The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998.