An Overview of the International Planning Competition Part 1: AI Planning and the IPC

Amanda Coles[1] Andrew Coles[1]
Álvaro Torralba[2] Florian Pommerening[3]

[1]King's College London

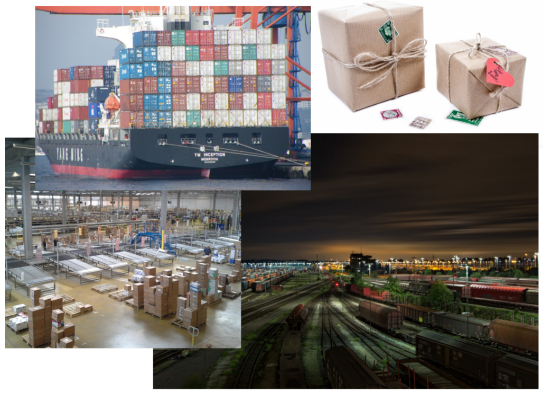
[2]Saarland University

[3]University of Basel, Switzerland

January 27, 2019

We have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 730086 and from the International Conference on Automated Planning and Scheduling.

Classical and Temporal Planning in a Nutshell



Classical planning

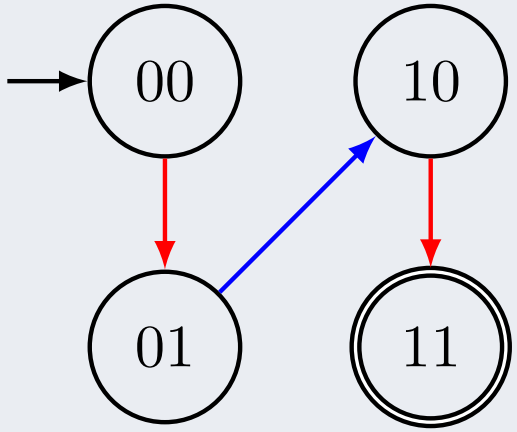
- Find operator sequence to achieve a goal
- Discrete, single-agent, observable, deterministic

Temporal planning

- actions take time and can be executed in parallel
- usually also includes numeric effects

Classical Planning Tasks

Example task: binary counter



State space

- States ○ assign values to variables
- Initial state →○
- Goal states ◎
- Operators ⟹ have conditions and effects on variables

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Compact Representation with PDDL

Domain

```
(define (domain trucks-example)
  (:requirements :typing)
  (:types truck location)
  (:predicates
    (CONNECTED ?from ?to - location)
    (truck-at ?t - truck ?l - location)
  )
  (:action move
    :parameters
      (?t - truck ?from ?to - location)
    :precondition
      (and (CONNECTED ?from ?to)
           (truck-at ?t ?from))
    :effect
      (and (not (truck-at ?t ?from))
           (truck-at ?t ?to))
  )
)
```

Task

```
(define (problem task1)
  (:domain trucks-example)
  (:objects
    t1 t2 - truck
    l1 l2 l3 - location
  )
  (:init
    (CONNECTED l1 l2)
    (CONNECTED l2 l3)
    (truck-at t1 l1)
    (truck-at t2 l3)
  )
  (:goal
    (and (truck-at t1 l3)
         (truck-at t2 l1))
  )
)
```

Evolution of PDDL

- 1998, 2000: PDDL 1.0: STRIPS, ADL (quantified effects and preconditions, conditional effects)
- 2002: PDDL 2.1: temporal + numeric planning
- 2004: PDDL 2.2: derived predicates and timed initial literals
- 2006: PDDL 3.0: soft goals and state trajectory constraints
- 2008: Restricted PDDL Features: STRIPS + action costs
- 2014: Re-introduced conditional effects

Solving Classical Planning Tasks

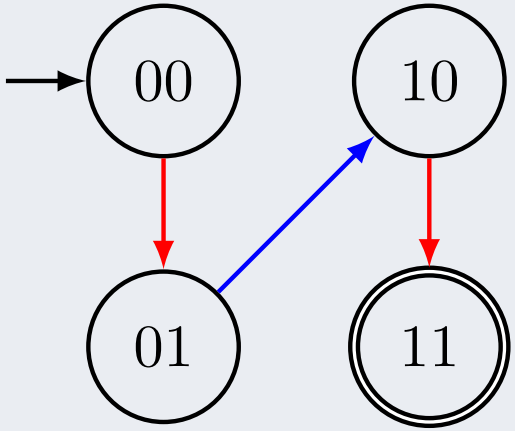
Search

Two important approaches

- explicit state search (A*, GBFS, ...)
  - every search node represents a state
  - expansion: generating successors for applicable operators
  - search guided by heuristic
- symbolic seach
  - every search node represents a set of states
  - expansion: generating all states reachable in one step
  - sets of states compactly represented (BDD, ...)
  - can also be guided by heuristic

Abstractions

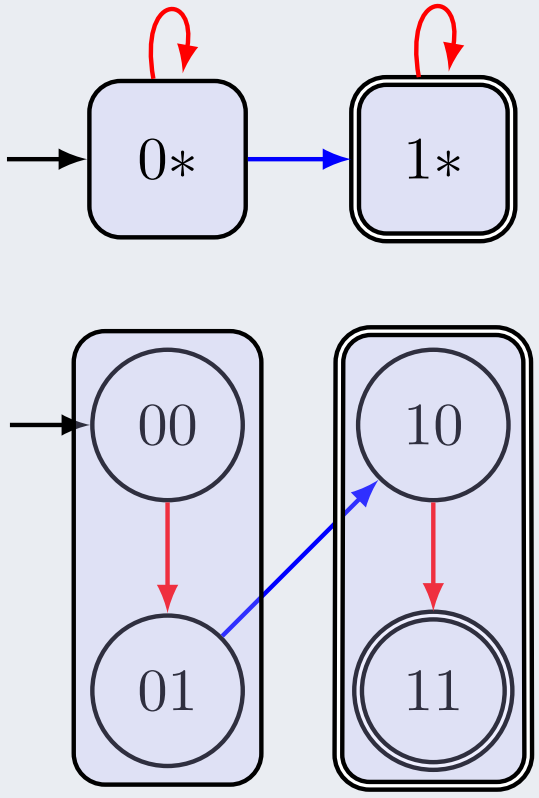
Abstractions of Planning Tasks



- full state space too big
  - example: plan for 10 trucks in 10 cities
- map to smaller space
- extract lower bound from abstractions

Abstractions

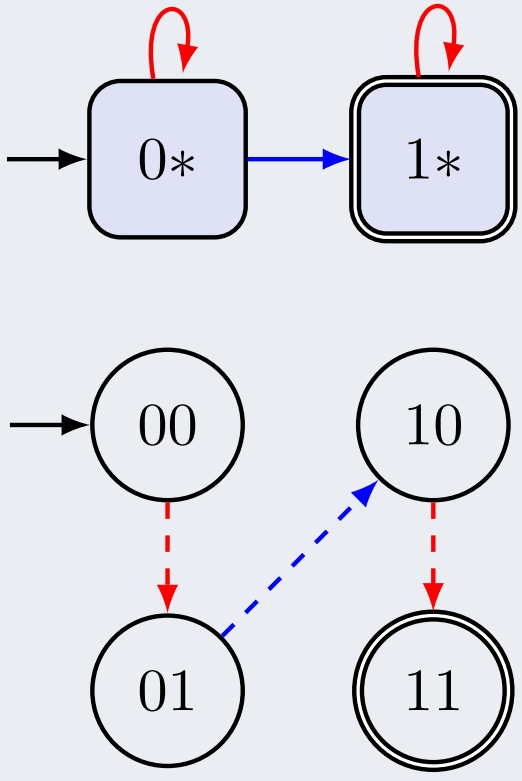
Abstractions of Planning Tasks



- full state space too big
  - example: plan for 10 trucks in 10 cities
- map to smaller space
- extract lower bound from abstractions

Abstractions

Abstractions of Planning Tasks



- full state space too big
  - example: plan for 10 trucks in 10 cities
- map to smaller space
- extract lower bound from abstractions

Delete Relaxations

Domain

```
(:action move
  :parameters
    (?t - truck ?from ?to - location)
  :precondition
    (and (CONNECTED ?from ?to)
         (truck-at ?t ?from))
  :effect
    (and (not (truck-at ?t ?from))
         (truck-at ?t ?to))
)
```

Delete Relaxations

- modify domain so deleting a fact never helps
- ignore some or all delete effects
- problem is simpler to solve
- heuristic value: solution cost in the relaxation

Delete Relaxations

Domain

```
(:action move
  :parameters
    (?t - truck ?from ?to - location)
  :precondition
    (and (CONNECTED ?from ?to)
         (truck-at ?t ?from))
  :effect
    (and (not (truck-at ?t ?from))
         (truck-at ?t ?to))
)
```

Delete Relaxations

- modify domain so deleting a fact never helps
- ignore some or all delete effects
- problem is simpler to solve
- heuristic value: solution cost in the relaxation

Delete Relaxations

Domain

```
(:action move
  :parameters
    (?t - truck ?from ?to - location)
  :precondition
    (and (CONNECTED ?from ?to)
         (truck-at ?t ?from))
  :effect
    (and ~~(not (truck-at ?t ?from))~~
         (truck-at ?t ?to))
)
```

Delete Relaxations

- modify domain so deleting a fact never helps
- ignore some or all delete effects
- problem is simpler to solve
- heuristic value: solution cost in the relaxation

Novelty

Novelty

- when exploring the state space prefer new areas
- a state is novel if we see parts of it for the first time
- the more general the part, the more novel the state
- limit search to only explore novel states
- can be combined with heuristics (best-first width search)

The International Planning Competition

IPC

The International Planning Competition (IPC)

- semi-regular competition
  - 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018
- organized in the context of the International Conference on Planning and Scheduling (ICAPS)

Past and Future IPCs

- `icaps-conference.org/index.php/Main/Competitions`
- `icaps-conference@googlegroups.com`

Goals of the IPC

Goals

- evaluate state-of-the-art planning systems
- promote planning research
- highlight challenges
- provide new benchmarks

IPC Organization

Organization

- different tracks for different planning variants
- tracks organized more or less independently
  - initiative of track organizers
  - IPC happens if someone organizes a track

Organizing a Track

Jobs as an organizer

- track rules
- benchmarks
  - create/elicit new domains
  - select instances
  - find reference solutions
- participants
  - elicit participation
  - compile planners
  - assist in testing/bug fixing
- experiments
  - run planners on benchmarks
  - evaluate results

IPC Tracks

Classical Planning Tracks

- satisficing (1998, 2000, 2004, 2006, 2008, 2011, 2014, 2018)
- optimal (2004, 2006, 2008, 2011, 2014, 2018)
- satisficing multi-core (2011, 2014)
- agile (2014, 2018)
- cost-bounded (2018)

Temporal Metric Planning

- satisficing (2002, 2004, 2008, 2011, 2014, 2018)
- optimal (2006, 2008, 2014)
- agile (2018)

...

IPC Tracks (continued)

Probabilistic Planning

- MDP (2004, 2006, 2011, 2018)
- conformant (2006, 2008)
- POMDP (2011)
- FOND, NOND (2008)
- continuous (2014)

Preferences, Constraints, Net-benefit

- satisficing (2006, 2008, 2014)
- optimal (2008, 2014)

Learning (2008, 2011, 2014)
Unsolvability (2016)
Hand-Tailored, Domain-Specific tracks (1998, 2000, 2002)