Planning and Optimization

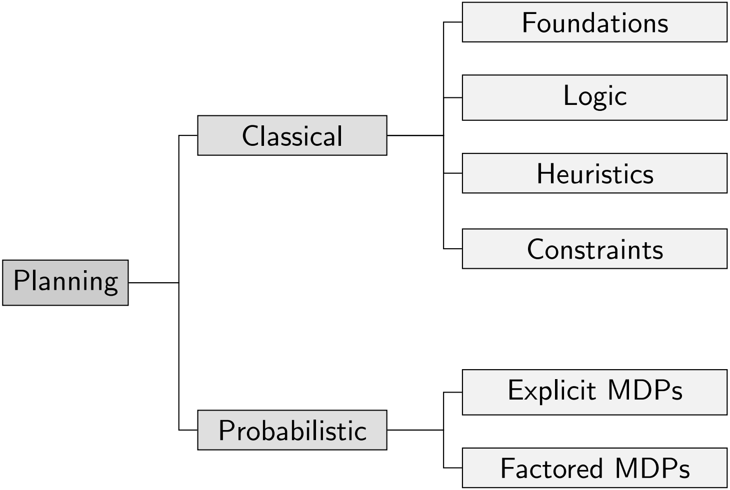
A2. What is Planning?

Malte Helmert and Gabriele Röger

Universität Basel

September 16, 2020

Content of this Course



Before We Start...

today: a very high-level introduction to planning

- our goal: give you a little feeling what planning is about
- preface to the actual course

$\rightsquigarrow$ main course content (beginning next week) will be mathematically formal and rigorous

- You can ignore this chapter when preparing for the exam.

Planning

General Problem Solving

Wikipedia: General Problem Solver

General Problem Solver (GPS) was a computer program created in 1959 by Herbert Simon, J.C. Shaw, and Allen Newell intended to work as a universal problem solver machine.

Any formalized symbolic problem can be solved, in principle, by GPS. [. . .]

GPS was the first computer program which separated its knowledge of problems (rules represented as input data) from its strategy of how to solve problems (a generic solver engine).

$\rightsquigarrow$ these days called “domain-independent automated planning”

$\rightsquigarrow$ this is what the course is about

So What is Domain-Independent Automated Planning?

Automated Planning (Pithy Definition)

"Planning is the art and practice of thinking before acting."

— Patrik Haslum

Automated Planning (More Technical Definition)

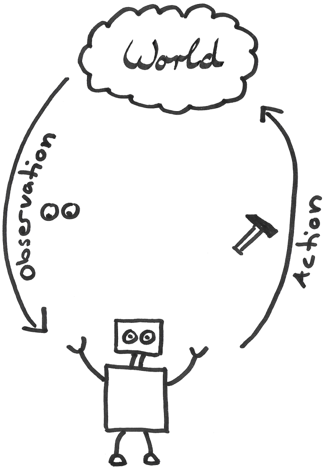
"Selecting a goal-leading course of action
based on a high-level description of the world."

— Jörg Hoffmann

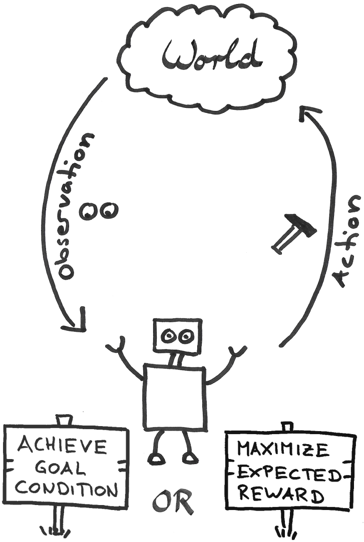
Domain-Independence of Automated Planning

Create one planning algorithm that performs sufficiently well
on many application domains (including future ones).

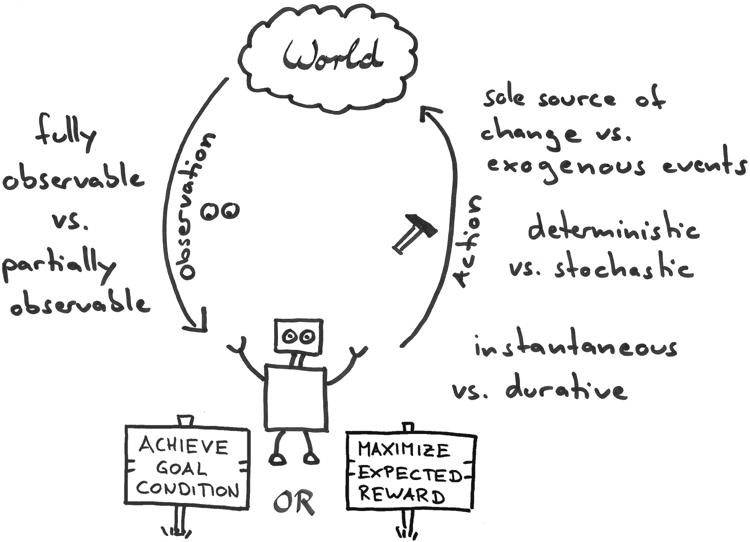
General Perspective on Planning



General Perspective on Planning



General Perspective on Planning



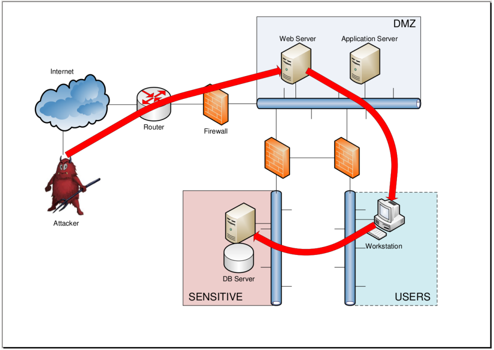
Example: Earth Observation

- satellite takes images of patches on Earth
- use weather forecast to optimize probability of high-quality images

Example: Termes

Harvard TERMES robots, based on termites

Example: Cybersecurity

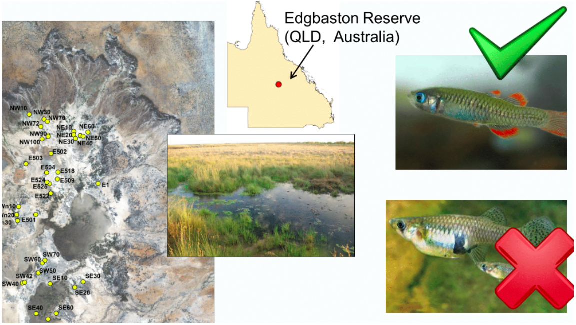


CALDERA automated adversary emulation system

Example: Intelligent Greenhouse

photo © LemnaTec GmbH

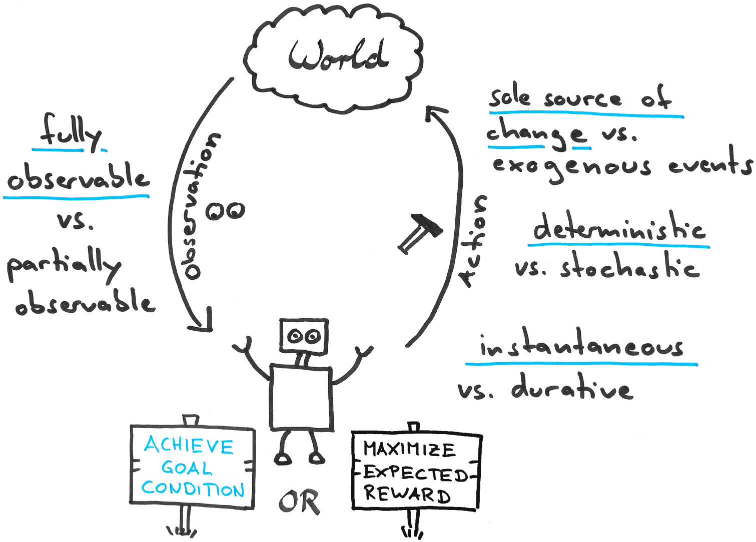
Example: Red-finned Blue-eye



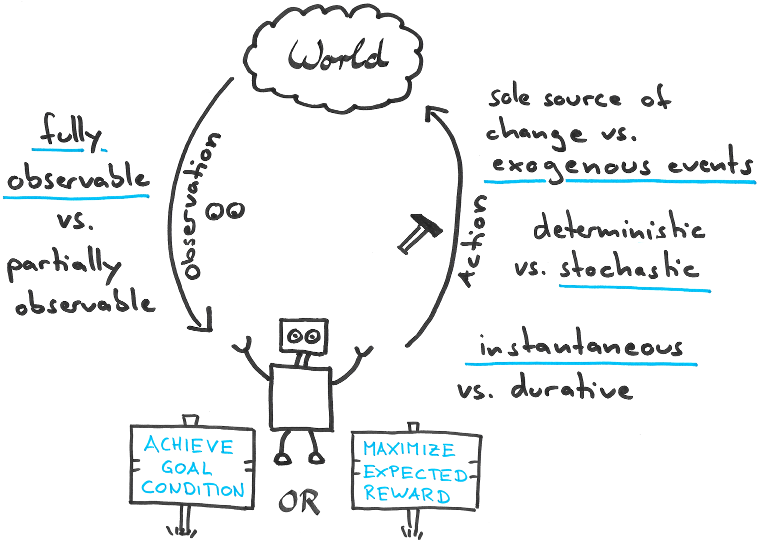
Picture by Iadine Chadès

- red-finned blue-eye population threatened by gambusia
- springs connected probabilistically during rain season
- find strategy to save red-finned blue-eye from extinction

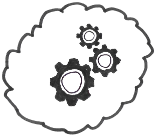
Classical Planning



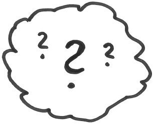
Probabilistic Planning



Model-based vs. Data-driven Approaches



Model-based approaches know the "inner workings" of the world
$\rightsquigarrow$ reasoning



Data-driven approaches rely only on collected data from a black-box world
$\rightsquigarrow$ learning

We focus on model-based approaches.

Planning Tasks

input to a planning algorithm: planning task

- initial state of the world
- actions that change the state
- goal to be achieved

output of a planning algorithm:

- plan (classical setting)
  - sequence of actions that takes initial state to a goal state
- policy (probabilistic setting)
  - function that returns for each state the action to take
- Why different concepts?

⇝ formal definitions later in the course

The Planning Research Landscape

- one of the major subfields of Artificial Intelligence (AI)
- represented at major AI conferences (IJCAI, AAAI, ECAI)
- annual specialized conference ICAPS ($\approx$ 250 participants)
- major journals: general AI journals (AIJ, JAIR)

Planning Task Examples

Example: The Seven Bridges of Königsberg

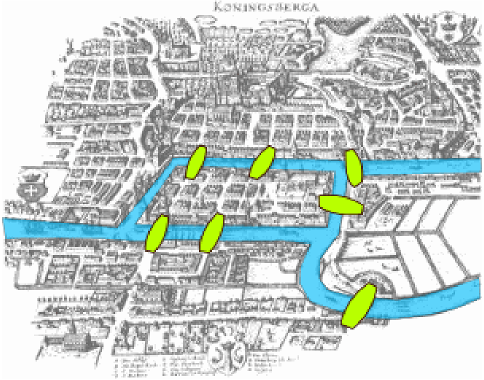


image credits: Bogdan Giuşcă (public domain)

Demo

```
$ ls demo/koenigsberg
```

Example: Intelligent Greenhouse

photo © LemnaTec GmbH

Demo

```
$ ls demo/ipc/scanalyzer-08-strips
```

Example: FreeCell

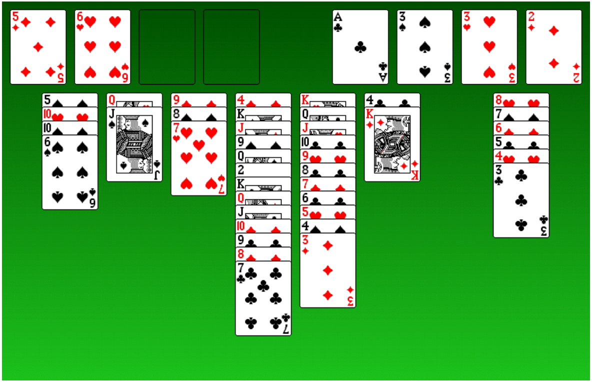


image credits: GNOME Project (GNU General Public License)

Demo Material

```
$ ls demo/ipc/freecell
```

Many More Examples

Demo

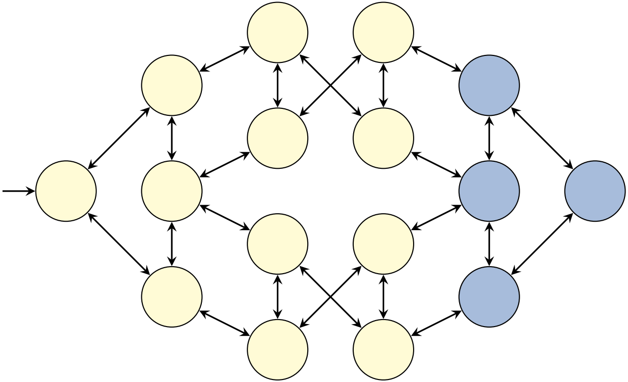
```
$ ls demo/ipc
agricola-opt18-strips
agricola-sat18-strips
airport
airport-adl
assembly
barman-mco14-strips
barman-opt11-strips
barman-opt14-strips
...
```

⇝ (most) benchmarks of planning competitions IPC 1998–2018

How Hard is Planning?

Classical Planning as State-Space Search

classical planning as state-space search:



⇝ much more on this later in the course

Is Planning Difficult?

Classical planning is computationally challenging:

- number of states grows exponentially with description size when using (propositional) logic-based representations
- provably hard (PSPACE-complete)

$\rightsquigarrow$ we prove this later in the course

Problem sizes:

- Seven Bridges of Königsberg: 64 reachable states
- Rubik's Cube: $4.325 \cdot 10^{19}$ reachable states
  $\rightsquigarrow$ consider 2 billion/second $\rightsquigarrow$ 1 billion years
- standard benchmarks: some with $> 10^{200}$ reachable states

Getting to Know a Classical Planner

Getting to Know a Planner

We now play around a bit with a planner and its input:

- look at problem formulation
- run a planner (= planning system/planning algorithm)
- validate plans found by the planner

Planner: Fast Downward

Fast Downward

We use the Fast Downward planner in this course

- because we know it well (developed by our research group)
- because it implements many search algorithms and heuristics
- because it is the classical planner most commonly used as a basis for other planners these days

$\rightsquigarrow$ http://www.fast-downward.org

Validator: VAL

VAL

We use the VAL plan validation tool (Fox, Howey & Long) to independently verify that the plans we generate are correct.

- very useful debugging tool
- `https://github.com/KCL-Planning/VAL`

Because of bugs/limitations of VAL, we will also occasionally use another validator called INVAL (by Patrik Haslum).

Illustrating Example: 15-Puzzle

9	2	12	7
5	6	14	13
3		11	1
15	4	10	8

$\longrightarrow$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Solving the 15-Puzzle

Demo

```
$ cd demo
$ less tile/puzzle.pddl
$ less tile/puzzle01.pddl
$ ./fast-downward.py \
      tile/puzzle.pddl tile/puzzle01.pddl \
      --heuristic "h=ff()" \
      --search "eager_greedy([h],preferred=[h])"
...
$ validate tile/puzzle.pddl tile/puzzle01.pddl \
      sas_plan
...
```

Variation: Weighted 15-Puzzle

Weighted 15-Puzzle:

- moving different tiles has different cost
- cost of moving tile x = number of prime factors of x

Demo

```
$ cd demo
$ meld tile/puzzle.pddl tile/weight.pddl
$ meld tile/puzzle01.pddl tile/weight01.pddl
$ ./fast-downward.py \
      tile/weight.pddl tile/weight01.pddl \
      --heuristic "h=ff()" \
      --search "eager_greedy([h],preferred=[h])"
...
```

Variation: Glued 15-Puzzle

Glued 15-Puzzle:

- some tiles are glued in place and cannot be moved

Demo

```
$ cd demo
$ meld tile/puzzle.pddl tile/glued.pddl
$ meld tile/puzzle01.pddl tile/glued01.pddl
$ ./fast-downward.py \
      tile/glued.pddl tile/glued01.pddl \
      --heuristic "h=cg()" \
      --search "eager_greedy([h],preferred=[h])"
...
```

Note: different heuristic used!

Variation: Cheating 15-Puzzle

Cheating 15-Puzzle:

- Can remove tiles from puzzle frame (creating more blanks) and reinsert tiles at any blank location.

Demo

```
$ cd demo
$ meld tile/puzzle.pddl tile/cheat.pddl
$ meld tile/puzzle01.pddl tile/cheat01.pddl
$ ./fast-downward.py \
      tile/cheat.pddl tile/cheat01.pddl \
      --heuristic "h=ff()" \
      --search "eager_greedy([h],preferred=[h])"
...
```

Summary

Summary

- planning = thinking before acting
- major subarea of Artificial Intelligence
- domain-independent planning = general problem solving
- classical planning = the "easy case"
  (deterministic, fully observable etc.)
- still hard enough!
  $\rightsquigarrow$ PSPACE-complete because of huge number of states
- probabilistic planning considers stochastic action outcomes and exogenous events.