Classical Planning Algorithms

1. What is Classical Planning?

Malte Helmert

ICAPS 2018 Summer School

June 21, 2018

About This Lecture

Lecture Materials and Demo

Lecture Materials

```
$ # Make sure your Vagrantfile is up to date.
$ vagrant up --provision
$ vagrant ssh
$ cd /vagrant/lectures/classical
$ ls demo
```

- You can omit the `--provision` flag if your VM is up to date.
- If you last updated it before Thursday, June 21, 12:00, the VM is not up to date.
- From now on, we assume you are inside the VM (`vagrant ssh`).

Goals of the Lecture

Goals of the Lecture

- find out what classical planning is
- get hands-on experience with planners and modeling
- get to know some core algorithmic ideas

Classical Planning: Today

Lecture Topics:

1. What is Classical Planning?
2. Planning Formalisms and Models
3. Planning Algorithms
4. Peeking Inside: Implementing a Simple Heuristic (time permitting)

Classical Planning: Tomorrow

more classical planning tomorrow:

- 9:00–10:30: Lecture Classical Planning Heuristics by Gabi Röger continues this lecture
- 16:00–open: Lab Classical and Probabilistic Planning

Classical 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"

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

$\rightsquigarrow$ formal definition in Part 2

Domain-Independence of Automated Planning

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

Classical Planning

This lecture covers classical planning:

- offline (static)
- discrete
- deterministic
- fully observable
- single-agent
- sequential (plans are action sequences)
- domain-independent

This is just one facet of planning.

More General Planning Topics

More general kinds of planning include:

- ~~offline~~: online planning; planning and execution
- ~~discrete~~: continuous planning (e.g., real-time/hybrid systems)
- ~~deterministic~~: probabilistic planning; FOND planning
- ~~single-agent~~: multi-agent planning; general game playing; game-theoretic planning
- ~~fully observable~~: POND planning; conformant planning
- ~~sequential~~: e.g., temporal planning

Classical planning techniques are often used as ingredients in more general kinds of planning.

Example: determinization in probabilistic planning

Planning Task Examples

Example: The Seven Bridges of Königsberg

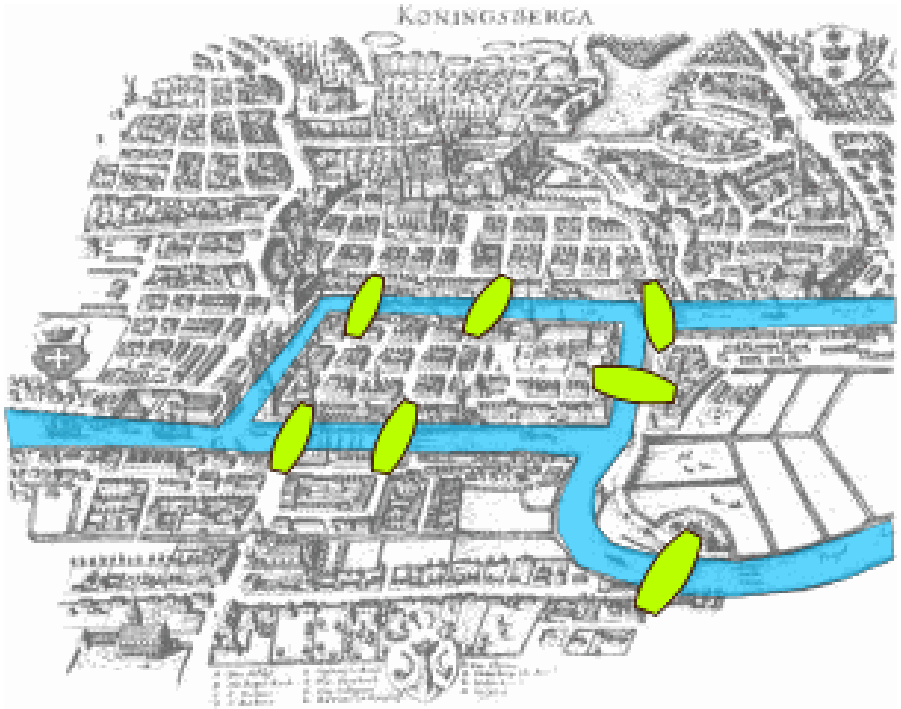


image credits: Bogdan Giușcă (public domain)

Demo Material

```
$ cd /vagrant/lectures/classical/demo
$ ls examples/koenigsberg
```

Example: Intelligent Greenhouse

photo © LemnaTec GmbH

Demo Material

```
$ cd /vagrant/lectures/classical/demo
$ ls examples/scanalyzer
```

Example: FreeCell

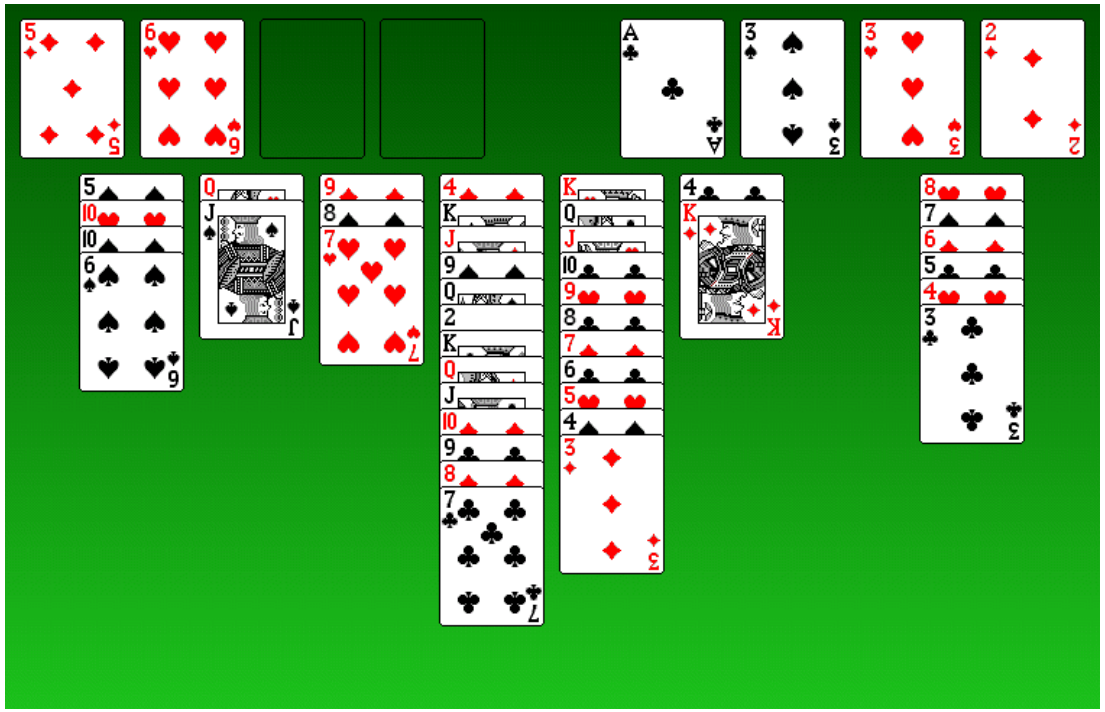


image credits: GNOME Project (GNU General Public License)

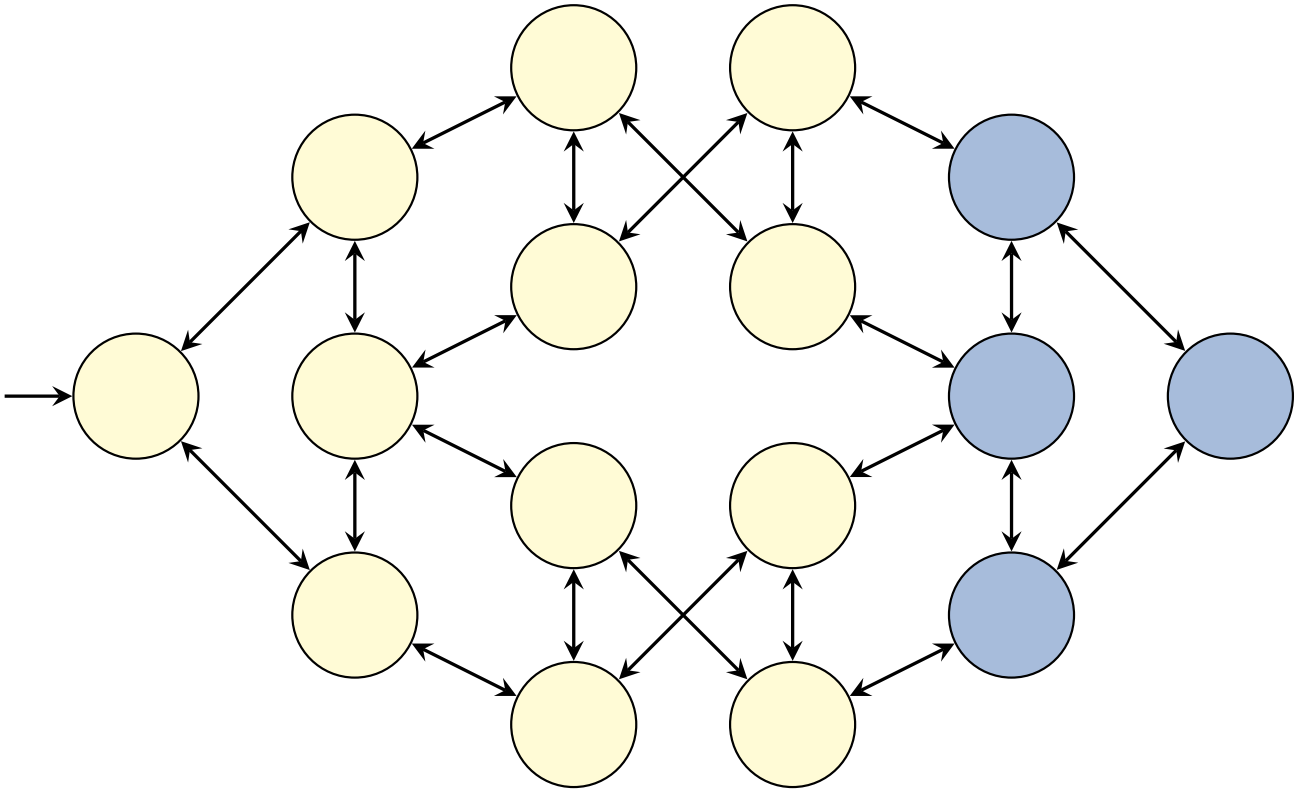
Demo Material

```
$ cd /vagrant/lectures/classical/demo
$ ls examples/freecell
```

How Hard is Planning?

Planning as State-Space Search

Planning as state-space search:



⇝ more in Part 3

Is Planning Difficult?

Classical planning is computationally challenging:

- number of states grows exponentially with description size when using "grounded" representations;
  doubly exponentially when using "schematic" representations
- provably hard (PSPACE-complete/EXPSPACE-complete)

Problem sizes:

- Seven Bridges of Königsberg: 64 reachable states
- Rubik's Cube: $4.325 \cdot 10^{19}$ reachable states
  $\rightsquigarrow$ consider 1 million/second $\rightsquigarrow$ 1.37 million years
- standard benchmarks: some with $> 10^{200}$ reachable states

Getting to Know a 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 for this summer school:

- because we know it well
- 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`

We emphasize that there are other great planners out there.

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
- can be used without a plan to diagnose PDDL errors

$\rightsquigarrow$ `https://github.com/KCL-Planning/VAL`

Illustrating Example: 15-Puzzle

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

→

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

Solving the 15-Puzzle

Demo

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

Note: The `./fd` shell script is an alias for
`/vagrant/fast-downward/fast-downward.py --build=release64`

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 /vagrant/lectures/classical/demo
$ meld tile/puzzle.pddl tile/weight.pddl
$ meld tile/puzzle01.pddl tile/weight01.pddl
$ ./fd 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 /vagrant/lectures/classical/demo
$ meld tile/puzzle.pddl tile/glued.pddl
$ meld tile/puzzle01.pddl tile/glued01.pddl
$ ./fd tile/glued.pddl tile/glued01.pddl \
      --heuristic "h=cg()" \
      --search "lazy_greedy([h],preferred=[h])"
...
```

Note: different search algorithm and 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 /vagrant/lectures/classical/demo
$ meld tile/puzzle.pddl tile/cheat.pddl
$ meld tile/puzzle01.pddl tile/cheat01.pddl
$ ./fd tile/cheat.pddl tile/cheat01.pddl \
      --heuristic "h=ff()" \
      --search "eager_greedy([h],preferred=[h])"
...
```

Summary

Summary

- planning = thinking before acting
- domain-independent planning = general problem solving
- classical planning = the "easy case"
  (deterministic, fully observable etc.)
- still hard enough! PSPACE-/EXPSPACE-complete
  because of huge number of states
- examples of planning tasks ($\rightsquigarrow$ demo material)