A Beginner's Introduction to Heuristic Search Planning

1. What is Planning?

Malte Helmert Gabriele Röger

AAAI 2015 Tutorial

January 25, 2015

About This Tutorial

Tutorial Materials

Tutorial Materials

http://ai.cs.unibas.ch/misc/tutorial_aaai2015/

About Us

Malte Helmert

Gabi Röger

Questions? Don't be shy to talk to us and/or email!

- `malte.helmert@unibas.ch`
- `gabriele.roeger@unibas.ch`

Target Audience

Target Audience

- everyone who is not yet an expert on planning and is interested in learning about it
- especially beginning PhD students and pre-PhD-level students
- we assume basic AI knowledge (e.g., a typical undergrad course) covering topics like A^* search
- but even without this, you should be able to follow

Please ask questions at any time!

Goals of the Tutorial and Topics Covered

Goals of the Tutorial

- learn about the planning research community
- find out how to become part of it
- get to know some core ideas
- get hands-on experience with planners and modelling

Three levels:

- automated planning: problem and research community
- classical planning: the "simplest" automated planning problem
- heuristics for solving classical planning tasks

More Details on Topics Covered

Automated planning:

- What is automated planning?
- Where is it studied?
- Where can I find out more?

Classical planning:

- What is classical planning?
- How can the problem be formalized?
- How can planning tasks be modelled?

Heuristics for solving classical planning tasks:

- What are heuristics and what is heuristic search?
- What are the major kinds of heuristics?
- Case studies of state-of-the-art heuristics

Table of Contents

Before the break:

After the break:

Demo and Hands-On

- Have you set up your hands-on directory/virtual machine?
- If yes, please start it up now and open a terminal window.
- If not, you can follow our demos and do the hands-on later. We will be happy to answer questions.

Hands-on instructions are given as follows:

Hands-On

```
$ cd hands-on
$ ./hello.sh
hello, tutorial!
$ ./build.sh
make:  Nothing to be done for ‘default’.
make:  Nothing to be done for ‘default’.
make:  ‘validate’ is up to date.
```

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).

The Planning Research Landscape

- one of the major subfields of artificial intelligence

$\rightsquigarrow$ represented at major AI conferences (IJCAI, AAAI, ECAI)

- annual specialized conference ICAPS
  - $\approx$ 200–250 participants
  - before 2003: ECP (odd years) + AIPS (even years)
- major journals: general AI journals (AIJ, JAIR)

Classical Planning

This tutorial 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.
Many others are studied in AI (at ICAPS and elsewhere).

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~~: FOND planning; probabilistic planning
- ~~single-agent~~: multi-agent planning; general game playing; game-theoretic planning
- ~~fully observable~~: POND planning; conformant planning
- ~~sequential~~: e.g., temporal planning

Domain-dependent planning problems in AI include:

- pathfinding, including grid-based and multi-agent (MAPF)
- continuous motion planning

Planning Task Examples

Example: The Seven Bridges of Königsberg

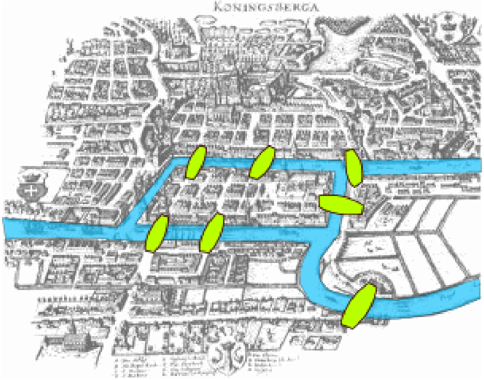


image credits: Bogdan Giușcă (public domain)

Hands-on Material

```
$ ls hands-on/koenigsberg
```

Example: Intelligent Greenhouse

photo © LemnaTec GmbH

Hands-on Material

```
$ ls hands-on/ipc/scanalyzer-08-strips
```

Example: FreeCell

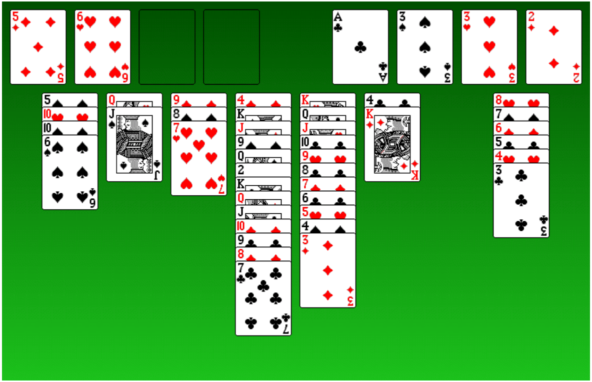


image credits: GNOME Project (GNU General Public License)

Hands-on Material

```
$ ls hands-on/ipc/freecell
```

Many More Examples

Hands-on Material

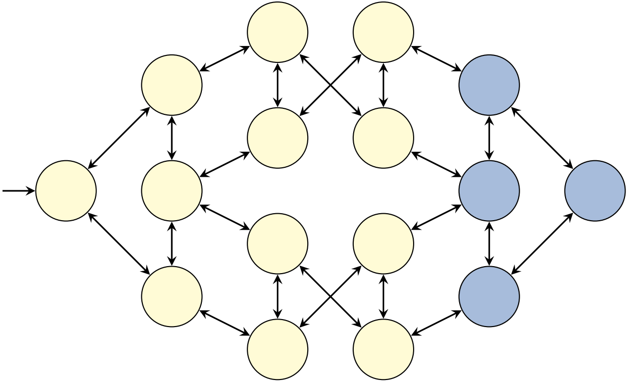
```
$ ls hands-on/ipc
airport
airport-adl
assembly
barman-opt11-strips
barman-sat11-strips
blocks
depot
driverlog
elevators-opt08-strips
...
```

$\rightsquigarrow$ (most) benchmarks of planning competitions IPC 1998–2011

How Hard is Planning?

Planning as State-Space Search

Planning as state-space search:



⇝ more in Part 2

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 2 billion/second $\rightsquigarrow$ 1 billion 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 our own Fast Downward planner for this tutorial:

- 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

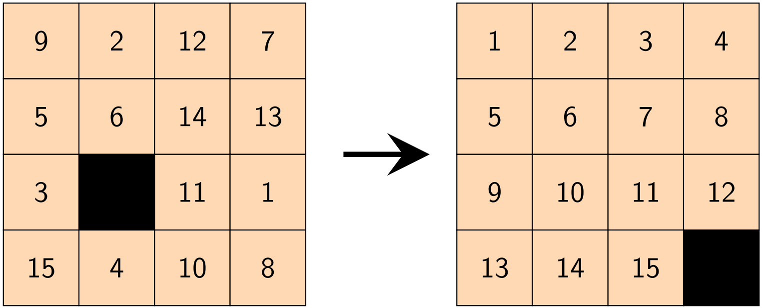
VAL

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

- very useful debugging tool
- included in Fast Downward repository

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

Illustrating Example: 15-Puzzle



Solving the 15-Puzzle

Hands-On

```
$ cd hands-on
$ 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
...
```

Variation: Weighted 15-Puzzle

Weighted 15-Puzzle:

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

Hands-On

```
$ cd hands-on
$ 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

Hands-On

```
$ cd hands-on
$ 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 "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.

Hands-On

```
$ cd hands-on
$ 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
- mainly studied at ICAPS (specialized), AAAI, IJCAI, ECAI
- 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
- many examples of planning tasks ($\leadsto$ hands-on material)
- tutorial focuses on one approach to classical planning, based on heuristic search