Planning and Optimization

M. Helmert, G. Röger
R. Christen, P. Ferber, T. Keller

University of Basel
Fall Semester 2022

Exercise Sheet 1

Due: October 03, 2022

Important: for submission, consult the rules at the end of the exercise. Non-adherence to these rules might lead to a penalty in the form of a deduction of marks or, in the worst case, that your submission will not be corrected at all.

Exercise 1.1 (2 marks)

Important: Unlike all other exercises in this course, this exercise is a prerequisite for exercises on upcoming exercise sheets.

The *Fast Downward* planning system is a tool that we use frequently for demos in the lecture and for exercises of the course. Your task in this exercise is to get access to the course repository as well as the planner installed and running. We describe two ways of setting up your system in the following: we start with the recommended way that uses *Vagrant* and *VirtualBox*, which should be possible on all Intel and AMD architectures. Afterwards, we point to relevant repositories and installation instructions in case you want to or have to set up your system manually. Please follow the instructions **in one** of these two possibilities.

Installation with *Vagrant* and *VirtualBox* Start by installing *Vagrant* and *VirtualBox* by following the installation instructions for your operating systems at `https://www.vagrantup.com` and `https://www.virtualbox.org`, respectively. If your operator system is Ubuntu 18.04 or 22.04, you can install both tools by running the following commands in a console:

```
$ sudo apt update
$ sudo apt install virtualbox vagrant (for Ubuntu 22.04)
$ sudo apt install virtualbox-qt vagrant (for Ubuntu 18.04)
```

With both tools installed, you can set up the virtual machine that runs the *Fast Downward* planner:

(a) Download the Vagrant configration file (*Vagrantfile*) from the website of the course.

(b) Copy or move the downloaded file to an *empty* directory. Make sure that your operating system didn't add a (possibly hidden) file extension (we have seen this happen frequently on Windows).

(c) Open a console in that directory and execute `$ vagrant up` (this may take a while).

Over the course of the semester, you'll have to interact with the virtual machine set up with *Vagrant* repeatedly. The most important commands to do so are:

- `$ vagrant up` to start the virtual machine (after the first time, this won't take as long)
- `$ vagrant halt` to stop the virtual machine
- `$ vagrant ssh` to connect to the virtual machine
- `$ vagrant exit` to disconnect from the virtual machine

.
You now have set up the virtual machine, cloned all relevant repositories and and installed required packages and tools. You have not yet compiled the Fast Downward planner that is used for this exercise, though. To do so, connect to the virtual machine (with `$ vagrant ssh`), then

(a) change to the directory with the *Fast Downward* version used for this exercise sheet with

```
$ cd /vagrant/planopt-hs22/exercise01/fast-downward
```

(b) compile the planner with `$ ./build.py`.

(c) run the planner on the first instance of the the Gripper benchmark domain with

```
$ ./fast-downward.py ../benchmarks/gripper/prob01.pddl --search "astar(blind())"
```

Congratulations, you have successfully set up your system for the practical part of the exercises! Now take a screenshot of the resulting console output of the run on the first Gripper instance and include it in your submission of this exercise sheet. There is no need to take multiple pictures if your screen doesn't fit the whole output; it is sufficient to provide an image that shows the "Actual search time" and all output following that line.

Manual Installation An alternative (which we do **not** recommend) is to install everything that is required manually. Start by cloning the following repositories:

- the repository of the course at `https://github.com/aibasel-teaching/planopt-hs22`
- the plan validator `val` at `https://github.com/KCL-Planning/VAL`
- the plan validator `INVAL` at `https://github.com/patrikhaslum/INVAL`

You can find the individual steps that are required to install both plan validators in the Vagrantfile. To give you an idea of what it happening in the Vagrantfile, please note that

- we are not using the latest version of `val`
- the `val` revision we use raises warnings during compilation that are treated as errors
- the produced binaries are moved to a folder on `PATH` (such that they can be executed from anywhere without providing a path to the binary)

If you manage to successfully install the validators, you still need to compile the Fast Downward planner that is used for this exercise sheet:

(a) Change to the directory with the *Fast Downward* version used for this exercise sheet. You can find it in the directory `exercise01/fast-downward` of the repository of the course.

(b) Follow the instructions on how to compile *Fast Downward* at `https://www.fast-downward.org/ObtainingAndRunningFastDownward`.

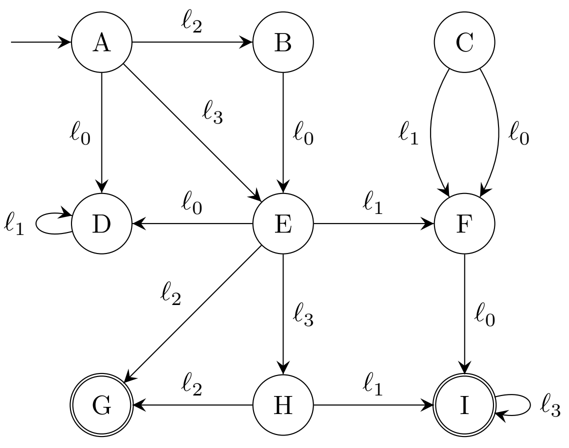
When you have successfully compiled *Fast Downward*, run the planner of the first instance of the Gripper benchmark domain with

```
$ ./fast-downward.py ../benchmarks/gripper/prob01.pddl --search "astar(blind())"
```

Take a screenshot of the resulting console output of the run and include it in your submission of this exercise sheet. There is no need to take multiple pictures if your screen doesn't fit the whole output; it is sufficient to provide an image that shows the "Actual search time" and all output following that line.

Exercise 1.2 (3+2 marks)

Consider the following transition system:



$$c(\ell_0) = 0, c(\ell_1) = 1, c(\ell_2) = 2, c(\ell_3) = 3$$

(a) Formalize the depicted transition system as a 6-tuple $\mathcal{T} = \langle S, L, c, T, s_0, S_\star \rangle$.

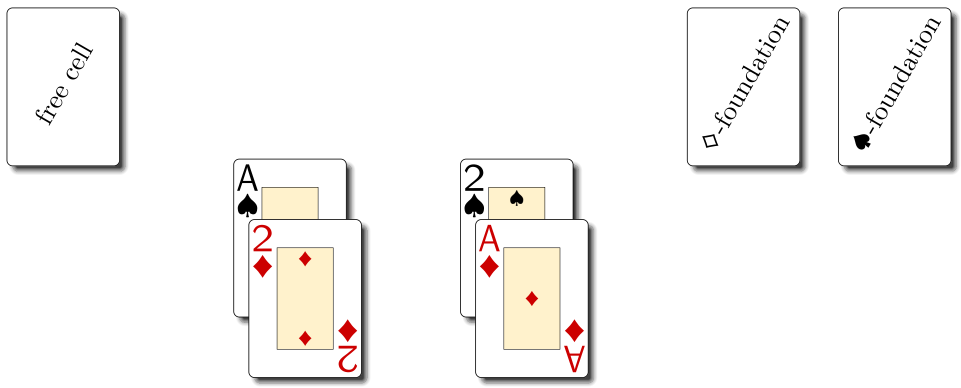
Hint: When writing down something formally, always ask yourself what kind of mathemtical object you are using where, then use correct notation. For instance, a tuple such as T is formalized using "angle" ($\langle\rangle$) or "round" ($()$) brackets; a set such as S is using braces ($\{\}$); a function such as $c : L \mapsto \mathbb{R}_0^+$ describes how objects in L are mapped to an object in $\mathbb{R}_0^+$; and a single object such as s_0 is using no brackets at all.

(b) Write down true statements (in plain english, not formally) about the depicted transition system using all terms from the following list. You may use several terms in the same sentence. The goal of this exercise is to confirm that you understand the meaning of these terms.

- *unreachable*
- *solution*
- *path length*
- *path cost*

Exercise 1.3 (3 marks)

ForcedFreeCell is a variant of the single player game FreeCell with identical rules (see `https://en.wikipedia.org/wiki/FreeCell` for the rules) except that cards that can be moved to the foundation may not be moved anywhere else. Graphically depict the transition system that consists of all states that are reachable from the following initial state:



Make sure that the direction of the transitions is clear and give a clear description of how to read your state. You may choose any representation for states. In particular, you may use a textual representation of states instead of actually drawing states as in the given figure. You may also skip redundant information as long as it is clear how to reconstruct the complete state from the provided information (e.g., you may omit all cards that are already placed in the correct foundation as long as you make clear that you do so).

Submission rules:

- Exercise sheets must be submitted in groups of three students. Please submit a single copy of the exercises per group (only one member of the group does the submission).
- Create a single PDF file (ending .pdf) for all non-programming exercises. Use a file name that does not contain any spaces or special characters other than the underscore "_". If you want to submit handwritten solutions, include their scans in the single PDF. Make sure it is in a reasonable resolution so that it is readable, but ensure at the same time that the PDF size is not astronomically large. Put the names of all group members on top of the first page. Either use page numbers on all pages or put your names on each page. Make sure your PDF has size A4 (fits the page size if printed on A4).
- For programming exercises, only create those code textfiles required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it. Code that does not compile or which we cannot successfully execute will not be graded.
- For the submission: if the exercise sheet does not include programming exercises, simply upload the single PDF. If the exercise sheet includes programming exercises, upload a ZIP file (ending .zip, .tar.gz or .tgz; *not* .rar or anything else) containing the single PDF and the code textfile(s) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly. After creating your zip file and before submitting it, open the file and verify that it complies with these requirements.
- Do not upload several versions to ADAM, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.