Planning and Optimization

M. Helmert, G. Röger
P. Ferber, T. Keller, S. Sievers

University of Basel
Fall Semester 2020

Exercise Sheet B

Due: October 18, 2020

Important: for submission, consult the rules at the end of the exercise. Non-adherence to the rules will lead to your submission not being corrected.

Exercise B.1 (8 marks)(Lecture B1)

Look up the following 5 planners in planner abstracts of the International Planning Competition (IPC) 2014 and 2018. Categorize each of them in a similar fashion as the examples in lecture B1. That is, list their problem class (satisficing or optimal), algorithm class (explicit search, SAT planning or symbolic search), the design choices of their respective class (for example the search direction for explicit search) and other aspects that stand out.

1. Mercury
2. MAPlan
3. Delfi
4. Madagascar
5. Symple

You can find planner abstracts on the competition websites reachable from the ICAPS website (`https://www.icaps-conference.org/competitions/`).

Exercise B.2 (3+3+3 marks)(Lecture B3)

(a) Consider the propositional planning task $\Pi = \langle V, I, O, \gamma \rangle$ with

$$
\begin{aligned}
V &= \{a, b, c, d, e\} \\
I(a) &= \mathbf{T} \\
I(v) &= \mathbf{F} \quad \text{for all } v \in V \setminus \{a\} \\
O &= \{o_1, o_2, o_3, o_4\} \\
\gamma &= e
\end{aligned}
$$

and

$$
\begin{aligned}
o_1 &= \langle \top, b \wedge d \rangle \\
o_2 &= \langle \neg e, a \wedge \neg b \rangle \\
o_3 &= \langle c, (d \rhd e) \rangle \\
o_4 &= \langle a \wedge \neg b, c \wedge \neg a \rangle
\end{aligned}
$$

Plot the search space explored by a progression and by a regression breadth-first search through this task. In the regression search simplify the state formula as much as possible at every node of the search tree. Do not expand the node further if that formula is unsatisfiable or logically entails the state formula of a previously expanded node. In the progression search do not expand a node if its state is a duplicate of a previously expanded state.

(b) Provide a family of planning tasks Π_n such that the size of Π_n is polynomial in n, and such that a breadth-first search with regression expands only a polynomial number of search nodes in n, whereas a breadth-first search with progression needs to expand an exponential number of search nodes in n. Assume the progression search prunes all duplicate states and the regression prunes a state if its formula logically entails the formula of its parent.

(c) Provide a family of planning tasks Π_n such that the size of Π_n is polynomial in n, and such that a breadth-first search with progression expands only a polynomial number of search nodes in n, whereas a breadth-first search with regression needs to expand an exponential number of search nodes in n. Assume the same pruning as in exercise (b).

Exercise B.3 (1+4+1+1)(Lecture B5)

In this exercise, your task is to define a SAT-encoding for the planning task $\langle\{a, b, c\}, \{a \mapsto 1, b \mapsto 0, c \mapsto 0\}, \{o_1, o_2\}, \neg a \wedge c\rangle$ with $o_1 = \langle a, b \wedge (b \rhd \neg a)\rangle$ and $o_2 = \langle a \wedge b, c\rangle$.

(a) Provide the clauses that encode the initial state and the goal (use time T for the latter).

(b) Provide all clauses that encode the transitions for some time step i. Simplify the clauses and omit those that simplify to $\top$ (you don't need to provide intermediate results for the simplification). Annotate each remaining clause as precondition clause, positive or negative effect clause or positive or negative frame clause.

(c) The clauses from Exercises B.3a) and B.3b) do not suffice for a complete SAT-encoding of the planning task. Provide all missing clauses, again parametrized for some time step i.

(d) What is the smallest horizon T for which the resulting formula for the given planning task is satisfiable? Justify your answer.

Exercise B.4 (3+5 marks)(Lecture B6)

For this exercise, you need to have minisat (`minisat.se/MiniSat.html`) installed. The simplest option is to install the package pysat, which will also install several other SAT solvers. You can install everything you need by running the following command: `./install-pysat.sh`

(a) The file `pyperplan/src/search/sat.py` already contains a complete implementation of a SAT search using a sequential encoding. Comment out the lines that add the positive frame clauses to the set of clauses. Explain why this is possible without making the SAT search compute incorrect solutions. Furthermore, investigate what effect on performance this change has experimentally. To do so, compare the runtime of the program with and without these clauses on the tasks in the directories `blocks`, `gripper` and `logistics`. You don't have to run the search longer than one minute.

You can run the code with the command
`./pyperplan/src/pyperplan.py -s sat-seq gripper/prob01.pddl`
(The domain file will be automatically inferred.)

Please note that the wallclock time printed by pyperplan can be quite off. Instead, please use the linux built-in `time` command by prepending it to the above command to obtain system wallclock time in seconds (example output: "real 0m32,177s").

(b) The file `pyperplan/src/search/sat.py` contains an incomplete method `build_parallel_model`. Please complete the implementation using the parallel encoding presented in the lecture. You don't have to change any of the other existing methods for this task.

Test your implementation on the same tasks as in part (a), using the command
`./pyperplan/ src/pyperplan.py -s sat-par gripper/prob01.pddl`.
What is the effect of the parallel encoding compared to the sequential one that you used in part (a)? Plase explain the reason for this effect.

Exercise B.5 (5+3 marks)(Lecture B8)

Pyperplan (`https://github.com/aibasel/pyperplan`) is a lightweight STRIPS planner written in Python. While it doesn't come with as strong performance as Fast Downward, it is very easy to extend and modify.

(a) In the file `pyperplan/src/search/bdd_bfs.py` you can find an incomplete implementation of a BDD-based breadth-first search. Complete it by using the utility methods in the file `pyperplan/src/search/bdd.py`. Do not modify anything else than the file `pyperplan/src/search/bdd_bfs.py` (and do not modify the constructor of `BDDSearch` yet, this is for part (b)). Test your search on the tasks in the directory `blocks` and make sure that it can find valid plans.

You can run the code with the command
`./pyperplan/src/pyperplan.py -s bdd blocks/domain.pddl blocks/p1.pddl`

(b) The constructor of `BDDSearch` contains a commented out alternative variable order for the variables within the BDD. Change the order by commenting out the old order and including the new order instead. Print the number of total BDD nodes after adding each operator and after each expansion step (use the provided method `print_bdd_nodes()`). Compare the two variable orders on a small task and discuss the results.

Submission rules:

- Exercise sheets must be submitted in groups of two or three students. Please submit one 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. If you want to submit handwritten parts, include their scans in the single PDF in a reasonable resolution, so that they are readable but the PDF size is not astronomically large. Put the names of all group members on top of the first page. Use page numbers 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 and submit 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!
- For the submission, you can either upload the single PDF or prepare 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. Please do not use directories within the ZIP, i.e., zip the files directly.
- Name all files without spaces.
- Only upload one submission per group. Do not upload several versions, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.